

AD-A252 173



NAVAL POSTGRADUATE SCHOOL
Monterey, California

2



DTIC
ELECTE
JUN 29 1992
S B D

THESIS

**FITTING AND PREDICTION UNCERTAINTY
FOR A
SOFTWARE RELIABILITY MODEL**

by

Thomas E. Dennison

March 1992

Thesis Advisor:

Donald P. Gaver

Approved for public release; distribution is unlimited.

92 6 20 02:0

92-16890

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL OR	7a. NAME OF MONITORING ORGANIZATION			
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code)			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Including Security Classification) FITTING AND PREDICTION UNCERTAINTY FOR A SOFTWARE RELIABILITY MODEL					
12 PERSONAL AUTHOR(S) DENNISON, Thomas E.					
13 TYPE OF REPORT Master's thesis	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day) 1992, March		15. Page Count 70	
16. SUPPLEMENTAL NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Software, Reliability, Bootstrap, Prediction Analysis, Software Reliability Models, Bayesian Methodology		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>The cost of system operational testing is steadily increasing. It is desirable for the software manager to know if the software is sufficiently well developed or reliable to support such testing. Current software reliability models provide only point estimates of the mean time to next failure or expected number of errors to occur in additional testing time.</p> <p>The goal of this thesis is to take into account prediction uncertainties of a software reliability model. Bootstrapping is used to provide the software manager with confidence limites of the predicted expected number of faults to occur for additional testing time. The results can be particularly useful to a software manager who has to answer a subjective question: is the software reliable enough to support system operational testing? A range of predicted expected number of faults will be of more use to a software manager, who has to justify the answer to this question, than just a point estimate. Two software fault data sets are analyzed with this techniques emphasizing how a software manager should analyze the results.</p>					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC			1a. REPORT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Donald P. Gaver			22b. TELEPHONE (Include Area Code) (408)646-2605	22c. OFFICE SYMBOL OR/Gv	

Approved for public release; distribution is unlimited.

Fitting and Prediction Uncertainty
for a
Software Reliability Model

by

Thomas E. Dennison
Lieutenant, United States Navy
B.S. Chemistry, Villanova University

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

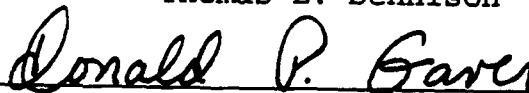
NAVAL POSTGRADUATE SCHOOL
March 1992

Author:



Thomas E. Dennison

Approved by:



Donald P. Gaver, Thesis Advisor



Timothy J. Shimeall, Second Reader



Peter Purdue, Chairman
Department of Operations Analysis

ABSTRACT

The cost of system operational testing is steadily increasing. It is desirable for the software manager to know if the software is sufficiently well developed or reliable to support such testing. Current software reliability models provide only point estimates of the mean time to next failure or expected number of errors to occur in additional testing time.

The goal of this thesis is to take into account prediction uncertainties of a software reliability model. Bootstrapping is used to provide the software manager with confidence limits of the predicted expected number of faults to occur for additional testing time. The results can be particularly useful to a software manager who has to answer a subjective question: is the software reliable enough to support system operational testing? A range of predicted expected number of faults will be of more use to a software manager, who has to justify the answer to this question, than just a point estimate. Two software fault data sets are analyzed with this technique emphasizing how a software manager should analyze the results.



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	SURVEY OF SOFTWARE RELIABILITY METHODOLOGIES . .	7
A.	TIME BETWEEN ERRORS (TBE)	7
	1. Jelinski and Moranda Model	8
	2. Schick-Wolverton Model	9
	3. Geometric Model	11
	4. Use of Time Between Errors (TBE) Models . .	13
B.	FAULT COUNT MODELS	14
	1. Generalized Poisson Model	14
	2. Non-homogeneous Poisson Process Model . . .	16
	3. Schneidewind's Software Reliability Model .	17
	4. Use of Fault Count Models	21
C.	SOFTWARE RELIABILITY MODELS	21
III.	DATA ANALYSIS	24
A.	MODEL DEVELOPMENT	24
B.	BOOTSTRAP	28
C.	RESULTS	29
D.	USE OF RESULTS	30
E.	APPLICATION TO TWO DATA SETS	31

IV. CONCLUSION	43
APPENDIX	46
REFERENCES	59
BIBLIOGRAPHY	61
INITIAL DISTRIBUTION LIST	63

ACKNOWLEDGMENTS

I owe a significant amount of gratitude to my advisor, Professor Donald P. Gaver. His guidance and encouragement were paramount in the completion of this research. He broadened my horizons, not only in statistical methods but also in how to communicate ideas. It would have been very difficult to complete this research without his advise and expertise.

I am also grateful to my second reader, Professor Timothy J. Shimeall for his willingness to evaluate this research.

Finally, but by no means less important, I am deeply indebted to my wife, Janice, who at the same that this research was being conducted endured a difficult pregnancy, compensated for my slack in parental duties during the many days I spent away from home, and managed a career job of her own. There is absolutely no possible way I could have completed this research without her encouragement and uncompromising, loving support. She is a true blessing. To my children , Jennifer, Sean, and Ryan, I thank you for your smiling, cheery faces that lifted my spirits after many days away from home.

I. INTRODUCTION

Prior to costly operational testing of a system consisting of hardware and its embedded software, it would be highly desirable to know whether these two major components are sufficiently reliable to support such testing. Specifically, this is equivalent to asking whether the software has reached a state of maturity such that unforeseen faults (bugs, errors, system crashes, etc.) are not likely to occur during operational test of the entire system, or later, during a systemic mission.

Estimation of hardware reliability is relatively well-understood. Unfortunately, software reliability or maturity prediction is not as well understood at this time. The ANSI/IEEE definition of software reliability is the ability of a program to perform a required function under stated conditions for a stated period of time (IEEE, 1984). Since testing software has an associated cost whether it is in computer run time, labor costs, lost market share resulting from late delivery of a product or, in the case of military equipment, sacrificed range-testing time and aborted missions, there is a finite time allocated for testing and removal of faults (bugs). A moderate-sized program with 264 branches would have 2^{264} independent paths (greater than the estimated number of atoms in the universe). Obviously, it is infeasible

to test each path (Dalal and Mallows, 1988). Testing and debugging costs are estimated to range from 50% to 80% of the costs for development of a working version of software (Beizer, 1984). The constraints of a finite time period for testing and the cost of testing are excellent incentives for prompt and accurate determination of software reliability. Put in the form of a question: when can testing be stopped and the product delivered with a high level of confidence that the customer will be satisfied?

Software reliability estimation is based on the results of testing. Software testing can be broken down into four major categories: unit, integration, system and regression testing. Unit testing is usually done by the programmer in an informal manner. Integration testing is done in an orderly progression such that the software elements are combined and tested until the entire software package has been tested. System testing is integration of hardware and software to verify that the system meets specified requirements. Regression testing is retesting to detect faults that may have been introduced during program modification (Hernandez, 1989). One purpose of testing is to produce quantitative measures of software error-proneness after effort has been expended in the integration testing, system testing, and fault removal phases.

Software testing, a follow-on to hardware reliability prediction has been of considerable importance and interest from the mid-1960's to the present. The Navy's Operational

Test and Evaluation Force recently (January, 1992) held a symposium for DoD agencies to discuss and exchange ideas and methodologies on software testing and reliability. There are two basic differences between hardware and software reliability predictions. Hardware prediction usually assumes independence of failures, and, after some point, the reliability measuring process does not affect the failure rate. Software reliability prediction models should assume interdependence of unit failures, and that testing improves reliability. Removing a program fault or bug during developmental testing reduces the likelihood that a fault will become operative later in an operational setting that will cause a mission to abort. The software fault-prevalence and appearance prediction problem has been judged to be inherently more difficult than hardware reliability prediction (Beizer, 1984).

There are several software reliability models that will be discussed later. Beizer in his seminal work *Software System Testing and Quality Assurance* (Beizer, 1984) summed up the similarities of the models best.

1. Most models assume a fixed but unknown number of faults when testing.
2. Faults are universally assumed to be independent (some of the later models, Schneidewind's Software Reliability Model for example, do not necessarily make this assumption).
3. Most models assume perfect debugging. That is, the debugging process introduces no new faults. However, some of the later models take into account that not all

detected faults will be fixed, and that the debugging process itself may introduce new faults (Littlewood and Verrall 's Bayesian Reliability Growth Model takes into account imperfect debugging).

4. Most models assume that test time and calendar time are the same.

5. The models assume that failure rate is proportional to the faults remaining. This implicitly means that faults are assumed to cause single failures and each failure can be related to one failure.

6. The models assume path homogeneity. That is, data are entered randomly and such data uniformly exercise all code. This is in direct contradiction to the reality that the most paths cover a small percentage (say under 10%) of the code.

The difference between the models lies in the degree with which these assumptions hold true, i.e. the type of random process according to which the failures occur, and how data is fitted to the models (Beizer, 1984).

The models that are described in Chapter II do not necessarily perform well for all types of data. There is no "silver bullet" (Brooks, 1986) that will take on all comers successfully. One model may predict reliability well for one data source but not another. The users of the models must take into consideration the predictive quality of a model prior to basing decisions on the output of the model (Abdalla et al, 1986) and (Goel, 1985). One possible way to do this is to analyze the data using various models. The manager selects the model that demonstrates the best predictive qualities, i.e. the model that appears to best fit the data and provide

useful results. The choice is difficult because it is conducted in an atmosphere of uncertainty.

Our hypothesis is that software reliability can be predicted, but with error. It is important to take account of the variabilities and uncertainties that are inevitably present, at least those associated with sampling (finite data), the most serious errors may be associated with model choice, however. To test this hypothesis of predictability we analyze sources of fault (error or bug) data using a modification of the BELLCORE MODEL (Dalal and Mallows, 1988) to estimate the reliability of the particular software project and the quality of the prediction produced by the model. Parametric estimates are made by maximum likelihood but also by use of an approximate Bayesian technique. Error estimates are made by a re-sampling technique known as bootstrapping.

The parametric bootstrap technique was used in the aftermath of the *Challenger* disaster to analyze the O-rings that failed. Although the analysis was done on hardware the methodology that we propose in Chapter III and the appendix is similar. The analysis of the O-rings showed the bootstrap 90% confidence limits expected catastrophic failure rate of at least 13% at temperature of less than 31 degrees, but less than a 2% failure rate at temperatures above 60 degrees (Dalal et al, 1989). Had the NASA decision makers had this information available to them the consideration to postpone the launch may have been taken more seriously and the disaster

prevented. The analogy for the software manager to consider is the predicted number of faults to occur for some specified time acceptable. It is hoped that, the wrong decision will not have consequences as severe as the *Challenger* disaster. The techniques that we describe provide a quantitative tool for the software manager to substantiate the decision to schedule (postpone) system operational testing.

In Chapter II, we briefly describe several software reliability prediction models that have been proposed in order to provide a basis of understanding of the discussion. In Chapter III and the appendix, we present the model fitting procedure, the method used to determine the quality of the prediction, the resulting data obtained from the analysis, and methods to improve this methodology from the perspective of a software manager. In Chapter IV, our conclusions are provided and directions for future research are suggested.

II. SURVEY OF SOFTWARE RELIABILITY METHODOLOGIES

This survey is concerned with only two categories of software reliability models: those for time between errors (TBE), and for fault count (number of errors in a specified time).

A. TIME BETWEEN ERRORS (TBE)

TBE reliability assessments attempt to predict the mean time between failure (MTBF) of the i th failure based on that to the $(i-1)$ th failure. The TBE can be measured in either central processing unit (CPU) time or wall-clock time. Wall-clock time can be misleading: it can elapse regardless of whether or not the program is running. From this information the software manager can gain confidence that the software will exhibit the operational capability to complete its mission: to operate without failure for a mission time. A system that experiences multiple, severe software errors that prevent the system from completing its operational mission is not ready for costly live exercises as in operational testing. For example, a system that is supposed to detect, track and engage a missile during a scenario of five minutes' duration, but whose software experiences a severe fault every thirty seconds on average, is obviously not ready to conduct an expensive live exercise or actual mission. Here are some

models that attempt to predict (mean or average) time to failure.

1. Jelinski and Moranda Model

Jelinski and Moranda developed the "De-Eutrophication Model" (Moranda and Jelinski, 1972), (Farr, 1983). The assumptions are:

- The rate of fault detection is proportional to the current fault content of a program.
- All faults are equally likely to occur and are independent of each other.
- Each fault is of the same severity as any other fault.
- The fault rate remains constant over the interval between fault occurrences.
- The software is operated in a manner similar to anticipated operational usage.
- The faults are corrected instantly, without introduction of new faults into the program.

The hazard rate for the i th fault is

$$Z_i(t) = \theta [N - (i-1)] , \quad (2.1)$$

where: N = total number of faults initially in the system

i = i th fault to occur

θ = proportionality constant.

$X_i = t_i - t_{i-1}$ is the time between the i th and the $(i-1)$ st fault and is assumed to have an exponential distribution with rate

$Z_i(t_i)$:

$$f(X_i) = \theta [N - (i-1)] e^{(-\theta [N - (i-1)] X_i)} . \quad (2.2)$$

The likelihood function for the parameters θ and N is

$$L(X_1, \dots, X_n) = \prod_{i=1}^n \theta [N - (i-1)] e^{[-\theta(N - (i-1))X_i]} . \quad (2.3)$$

Taking the partial derivatives of $\ln(L)$ with respect to N (N is allowed to assume any real value as a convenient approximation) and θ , and then setting the equations equal to zero, the solutions for the following set of equations are obtained as maximum likelihood estimates for N and θ (N is estimated by numerical techniques, then used to solve for θ):

$$\theta = \frac{n}{\hat{N}(\sum_{i=1}^n X_i) - \sum_{i=1}^n (i-1)X_i} , \quad (2.4)$$

$$\sum_{i=1}^n \frac{1}{\hat{N} - (i-1)} = \frac{n}{\hat{N} - \frac{1}{\sum_{i=1}^n X_i} (\sum_{i=1}^n (i-1)X_i)} . \quad (2.5)$$

The estimate for the mean time between failure (MTBF) for the $(i+1)$ st fault occurrence is

$$MTBF_{i+1} = \frac{1}{Z(t_i)} = \frac{1}{\theta(\hat{N} - i)} . \quad (2.6)$$

The data required to use the Jelinski-Moranda model are the observed times of the fault occurrence (t_i 's), or the times between the faults (x_i 's).

2. Schick-Wolverton Model

The hazard rate for the Schick-Wolverton model (Schick and Wolverton, 1978) and (Farr, 1983) is proportional

to the number of faults in the program and the amount of testing time. An assumption of the model is that as more testing is completed the probability of detecting faults increases because of "zeroing-in" on the areas of code where the errors lie. The assumptions are:

- The rate of fault detection is proportional to the current fault content and to the amount of time expended in testing.
- All faults are equally likely to occur
- All faults are independent of each other
- All faults are of the same severity
- The software is operated in a manner similar to the anticipated operational usage
- Perfect fault correction occurs.

The hazard function is

$$Z(X_i) = \theta [N - (i-1)] X_i , \quad (2.7)$$

where: X_i = the amount of time spent testing between the occurrence of the i th and the $(i-1)$ st fault
 N = total number of faults initially in the program
 θ = proportionality constant.

The reliability function of X_i is

$$R(X_i) = \exp(-\theta [N - (i-1)] \frac{X_i^2}{2}) . \quad (2.8)$$

The density function of X_i is

$$f(X_i) = -R'(X_i) = \theta [N - (i-1)] X_i e^{(-\theta [N - (i-1)] \frac{X_i^2}{2})} \quad (2.9)$$

If $X_i^2/2$ is replaced by Y_i the model is formally identical to the Jelinski-Moranda model previously described. In fact, substitution of any known function of X_i allows transformation to the Jelinski-Moranda model. N and θ are estimated by MLE's:

$$\theta = \frac{2n}{\sum_{i=1}^n (\hat{N} - (i-1)) X_i^2} \quad (2.10)$$

$$\sum_{i=1}^n \frac{1}{(\hat{N} - (i-1))} = \theta \frac{\sum_{i=1}^n X_i^2}{2} \quad (2.11)$$

The estimate for the mean time between failure (MTBF) for the $(i+1)$ st occurrence is

$$MTBF_{i+1} = \sqrt{\frac{1}{2\theta (\hat{N} - i)}} \quad (2.12)$$

The data requirements are the time of the fault occurrence, t_i , or the time between the i th and $(i-1)$ st fault.

3. Geometric Model

The Geometric model (Moranda, 1975) and (Farr, 1983) is a modification of the Jelinski-Moranda "De-Eutrophication" model. It differs from that model as follows: it does not assume a fixed number of faults in the program, and the faults are not equally likely to occur because as debugging

progresses faults become harder to detect. The assumptions are:

- There is an infinite number of total faults (the program is never totally fault free).
- All faults do not have the same chance of detection.
- Detections of faults are independent.
- The software is operated in a manner similar to anticipated operational usage.
- The fault detection rate forms a geometric progression and is constant between faults.

The hazard rate for the i th fault is

$$Z_i(t) = D\theta^{i-1}, \quad (2.13)$$

where: t_i = time between the i th and the $(i-1)$ th fault

D = initial hazard rate

θ = fault detection rate ($0 < \theta < 1$)

n = the n th fault to occur.

X_i = time between the i th and the $(i-1)$ st fault. The X_i are independently and exponentially distributed with rate $Z_i(t)$, so the density function of X_i is

$$f(X_i) = D\theta^i e^{-D\theta^{i-1}X_i}, \quad (2.14)$$

D and θ are estimated by MLE's:

$$\hat{D} = \frac{\theta^n}{\sum_{i=1}^n \theta^i X_i}, \quad (2.15)$$

$$\frac{\sum_{i=1}^n i\theta^i X_i}{\sum_{i=1}^n \theta^i X_i} = \frac{n+1}{2} . \quad (2.16)$$

Equation (2.16) is solved for $\hat{\theta}$, and that value is substituted into (2.15) to find \hat{D} . From these equations the MTBF until the (n+1)st fault occurs after n faults have occurred can be obtained:

$$MTBF_{n+1} = \hat{E}(X_{n+1}) = \frac{1}{\hat{D} \hat{\theta}^n} . \quad (2.17)$$

The data requirements are the time of the ith fault (t_i), or the time between the faults (X_i), for $i = 1, 2, \dots, n$.

4. Use of Time Between Errors (TBE) Models

The TBE for models in this category can be measured in either wall-clock time or CPU time. The models may be used to predict the expected time to the next failure. Confidence limits on the expected value should be used to obtain a range of time to the next failure. The software manager should be asking: is the expected time of next time of failure longer than the time required for operational testing of the software within the overall system? If the time required for operational testing of the system is greater than the mean time to failure for the (i+1)th failure then the prudent software manager should consider postponing operational

testing in favor of continued developmental activity and testing.

B. FAULT COUNT MODELS

Fault count models use the number of faults that occurred in a testing interval to determine the expected number of faults in the next testing interval. Software managers can employ this method by simply counting the number of faults in a given test period i.e. day, week, or month, provided test exposures are the same. This provides insight into how well the testing process is working.

1. Generalized Poisson Model

The Generalized Poisson Model (Schafer et al, 1979), (Farr, 1983) is similar to the Jelinski-Moranda and Schick-Wolverton models but uses fault count observations in fixed, equal-length intervals rather than times between faults. The assumptions are:

- The expected number of faults occurring in any time interval is proportional to the fault content (number of bugs remaining) at the time of testing, and to the amount of time that has been previously spent in testing. The actual number of faults that appear is assumed to be Poisson distributed.
- All faults are equally likely to occur and are independent of each other.
- Each fault is of the same severity.
- The software is operated in a manner similar to the anticipated operational usage.

- The faults are corrected at the ends of the testing intervals. (Note: Faults discovered in one test interval may be corrected at another test interval; the only restriction is that the fault correction come at the end of the testing intervals.)

Testing intervals are of length x_i , and f_i faults occur during the i th interval. At the end of the i th interval a total of M_i faults are corrected.

The expected number of faults in the i th interval is

$$E(f_i) = \theta [N - M_{i-1}] g_i(x_1, x_2, \dots, x_i) , \quad (2.18)$$

where: θ = proportionality constant

N = initial number of faults

g_i = function of the amount of testing time spent previously and currently and is nondecreasing; as testing progresses more faults are found specifically,

$$g_i(x_1, x_2, \dots, x_i) = x_i^\alpha , \quad (2.19)$$

where α is assumed known.

f_i is Poisson with mean = $\theta (N - M_{i-1}) g_i$. N and θ are estimated by

MLE's:

$$\theta = \frac{\sum_{i=1}^n f_i}{\hat{N} \sum_{i=1}^n g_i - \sum_{i=1}^n M_{i-1} g_i} , \quad (3.20)$$

$$\sum_{i=1}^n \frac{f_i}{(\hat{N} - M_{i-1})} = \theta \sum_{i=1}^n g_i . \quad (2.21)$$

These non-linear equations must be solved for $\hat{\theta}$ and \hat{N} . From this the expected number of errors in the $(n+1)$ st test interval can be obtained,

$$\hat{E}(f_{n+1}) = \hat{\theta} (\hat{N} - M_n) g_{n+1}(x_1, \dots, x_{n+1}), \quad (2.22)$$

where: x_{n+1} is the anticipated testing time for the $(n+1)$ st test interval.

The data requirements for this model are the lengths of the test intervals, (x_i) , the total number of faults corrected at the end of a test interval, (M_i) , and the number of faults discovered in each interval (f_i) .

2. Non-homogeneous Poisson Process Model

The Non-homogeneous Poisson Process Model (NHPP) (Goel and Okumoto, 1979) and (Farr, 1983) assumes that the fault counts for testing intervals follows a Poisson distribution. The expected number of faults in the Poisson process model is proportional to the number of faults left in the program. The assumptions are:

- The software is operated in a manner similar to the anticipated operational usage.
- The numbers of faults detected, (f_i) , in the any test interval, (t_{i-1}, t_i) , are independent for any finite collection of times $t_1, < t_2, \dots, t_i, \dots, t_{m-1}, t_m$.
- Faults are of the same severity.
- Faults are equally likely to be detected.
- The cumulative number of faults detected at any time t , $(N(t))$, is a Poisson distribution with mean $m(t)$. The mean, $m(t)$, is the expected number of faults to occur for

any time period $(0, t)$ and is proportional to the expected number of undetected faults at time t .

- $m(t)$ is bounded.

The specific mean function used is

$$m(t) = a(1 - e^{-bt}) , \quad (2.23)$$

and f_i is the number of faults in the i th interval,

$$f_i = N(t_i) - N(t_{i-1}) , \quad (2.24)$$

where: a = expected total number of faults to be eventually detected.

a and b can be estimated by MLE's:

$$\hat{a} = \frac{\sum_{i=1}^m f_i}{(1 - e^{-\hat{b}t_m})} , \quad (2.25)$$

$$\frac{t_m e^{-\hat{b}t_m} \sum_{i=1}^m f_i}{(1 - e^{-\hat{b}t_m})} = \sum_{i=1}^m \frac{f_i (t_i e^{-\hat{b}t_i} - t_{i-1} e^{-\hat{b}t_{i-1}})}{e^{-\hat{b}t_{i-1}} - e^{-\hat{b}t_i}} . \quad (2.26)$$

From the estimates of a and b the expected number of faults in the next $(m+1)$ st test interval is estimated to be

$$m(t_{m+1}) - m(t_m) = \hat{a}(e^{-\hat{b}t_m} - e^{-\hat{b}t_{m+1}}) . \quad (2.27)$$

The data required for this model are the fault counts of each test interval, (f_i) and time of the test interval, (t_i) .

3. Schneidewind's Software Reliability Model

Schneidewind's model (Schneidewind, 1975) and (Farr, 1983) maintain that as testing progresses the fault detection

process changes. The later faults are therefore more useful in determining future fault counts. The model allows for three approaches.

1. Utilize all the fault counts from the m intervals.
2. The first $(s-1)$ intervals are ignored and only the s through m interval fault counts are considered.
3. The first $(s-1)$ intervals fault counts are summed, and the individual fault count from the remaining s through m intervals are treated individually. Denote the sum of the fault counts in the first $s-1$ intervals by:

$$F_{s-1} = \sum_{i=1}^{s-1} f_i . \quad (2.28)$$

Method 1 is used when the analyst feels that all intervals will be useful. Method 2 can be used when a significant change in the fault detection process has occurred at approximately the $(s-1)$ st interval. Method 3 attempts to combine the effects of both approaches. The assumptions for all methods are the same:

- The fault counts for each interval are independent of each other.
- The fault correction rate is proportional to the number of faults to be corrected.
- The software is operated in a manner similar to the anticipated operational usage.
- The mean number of detected faults decreases from one interval to the next.
- Intervals are all of the same length.

- The rate of fault detection is proportional to the number of faults remaining. The fault detection process is assumed to be a non-homogeneous Poisson process with an exponentially decreasing appearance and detection rate.

The rate of change of the number of faults detected in the i th interval is

$$d_i = \alpha e^{(-\beta i)} . \quad (2.29)$$

The cumulative mean number of faults that occurs up to and including interval i is

$$D_i = \frac{\alpha}{\beta} (1 - e^{-\beta i}) . \quad (2.30)$$

The mean number of faults for the i th interval is

$$m_i = D_i - D_{i-1} = \frac{\alpha}{\beta} (e^{(-\beta(i-1))} - e^{(-\beta i)}) . \quad (2.31)$$

α and β can be determined by MLE's:

$$\beta = \ln(y) , \quad (2.32)$$

$$\alpha = \frac{(\sum_{i=1}^m f_i) \beta}{1 - e^{-\beta m}} . \quad (2.33)$$

For Method 1, y is the solution to:

$$\frac{F_m}{y-1} - \frac{mF_m}{y^m-1} = A , \quad (2.34)$$

where:

$$A = \sum_{i=0}^{m-s} (s+i-1) f_{s+i} , \quad (2.35)$$

$$F_m = \sum_{i=1}^m f_i . \quad (2.36)$$

For Method 2, y is the solution to

$$Ay^{m-s+2} - (A+F_{s,m})y^{m-s+1} + ((m-s+1)F_{s,m}-A)y + (A+F_{s,m}-(m-s+1)F_{s,m}) = 0 , \quad (2.37)$$

where:

$$A = \sum_{i=0}^{m-s} i f_{s+i} , \quad (2.38)$$

$$F_{s,m} = \sum_{i=s}^m f_i , \quad (2.39)$$

$$\hat{\alpha} = \frac{(\sum_{i=s}^m f_i) \beta}{1 - e^{-\beta}} . \quad (2.40)$$

For Method 3, y is the solution to

$$\frac{(s-1)F_{s-1}}{y^{s-1}-1} + \frac{F_{s,m}}{y-1} - \frac{mF_m}{y^m-1} = A , \quad (2.41)$$

where: A is the same as Method 1 and $F_{s,m}$ is the same as Method

2. From the MLE's of $\hat{\alpha}$ and $\hat{\beta}$ the expected number of faults in

the $(m+1)$ st interval is

$$\hat{E}(f_{m+1}) = \frac{\hat{\alpha}}{\hat{\beta}} (e^{-\hat{\beta}} - e^{-\hat{\beta}(m+1)}) . \quad (2.42)$$

The time needed to detect a total number of M faults is

$$\frac{\log\left(\frac{\alpha}{(\alpha - \beta M)}\right)}{\beta} \quad (2.43)$$

The data needed for this model are the fault counts for each interval and a history of testing process in order to determine the interval that testing procedures may have altered significantly.

4. Use of Fault Count Models

Fault count models use the number of faults that occur in some testing interval. The models in this category predict the expected number of faults to occur in some additional time interval. Confidence limits on the expected number should be used to obtain a range of the predicted number of faults to occur for that time interval. Since there can never be a one hundred percent guarantee of perfect software, the software manager should be asking: is the predicted number of faults to occur for the time interval of interest acceptable for operational testing? If the predicted number of faults to occur is too great then the prudent software manager should postpone operational testing in favor of continued developmental activity and testing.

C. SOFTWARE RELIABILITY MODELS

The number of software reliability models continues to grow. Assumptions have broadened to reflect the reality of

the software development process with increased accuracy. The assumptions of some models described appear to be limiting. Faults all of the same severity can be worked around by modeling faults according to severity. The assumption that all faults are equally likely to occur and independent of each other can be resolved by assuming low severity faults occur more frequently than high severity faults, but faults of the same severity class will be considered equally likely to occur. Instantaneous fault correction can be avoided by not counting faults which were previously detected (and counted at time of initial detection), but were not corrected (Farr, 1983).

Software managers need to be aware of the limitations and underling assumptions that underlie the various models that are available. The data that is needed to fit the models is critical to reliable results. The data collection needs to be an accurate reflection of the meaningful historical testing of the software. Some of the data that should be collected is computer usage time, testing intensity, extent of the software that was tested (was the entire system tested or just a particular module), and milestones in the software's development (are requirements changed or added midway through the development of the software?) and, of course, the cost of testing.

This study illustrates the use of a particular reliability model. Some of the specific questions that this thesis

addresses are: How is a software reliability model used? What type of information does a model require? What kind of decision can a software manager make based on the results of the reliability model?

In today's fiscal environment software managers should have a "warm fuzzy feeling" substantiated by quantitative results for their product prior to initiating costly full scale, live operational testing.

III. DATA ANALYSIS

A. MODEL DEVELOPMENT

The model that is applied in this thesis is based on the assumption that the rate of error occurrence is a non-stationary Poisson process (NSPP) (Dalal and Mallows, 1988). The model is identical to the Schneidewind model, and is fitted according to Method 1, which assumes that all fault data is of equal value. Let $N(t)$ be the number of faults that occur in $(0, t)$; where t is software running time. The probability that the number of faults to occur by time t is given by:

$$P\{N(t)=n\} = \frac{e^{-\lambda(t)} (\lambda(t))^n}{n!}, \quad (3.1)$$

where $\lambda(t) = \lambda(1 - e^{-\mu t})$. A test time, t_1 , was chosen. This length of time is divided into periods of length $\Delta = t_1/J$; where J is the total number of intervals. The j th interval is such that $(j-1)\Delta < t < j\Delta$. The number of observed counts (faults) in the j th interval is n_j . The probability distribution for the number of faults in $[(j-1)\Delta$ to $j\Delta]$ is

$$P\{N_j = N(j\Delta) - N((j-1)\Delta) = n_j\} = e^{-\lambda_j} \frac{(\lambda_j)^{n_j}}{n_j!}, \quad \mu_j = 0, 1, 2, \quad (3.2)$$

where

$$\lambda_j = E[N_j] = \lambda (1 - e^{-\mu j \Delta}) - \lambda (1 - e^{-\mu (j-1) \Delta}) , \quad (3.2a)$$

$$= \lambda e^{-\mu (j-1) \Delta} (1 - e^{-\mu \Delta}) . \quad (3.2b)$$

The parameters μ and λ are estimated by maximum likelihood.

The likelihood function is

$$L(\lambda, \mu) = \prod_{j=1}^J e^{-\lambda_j} \frac{(\lambda_j)^{n_j}}{n_j!} . \quad (3.3)$$

The natural log of $L(\lambda, \mu)$ is

$$l(\lambda, \mu) = \ln(L) = -\sum_{j=1}^J \lambda_j + \sum_{j=1}^J n_j \ln(\lambda_j) . \quad (3.4)$$

The partial derivatives of $l(\lambda, \mu)$ with respect to λ and μ are taken and set equal to zero. This allows $\hat{\lambda}$ to be written in terms of μ and $n(t_s)$, the total number of counts to occur up to time t_s , as

$$\hat{\lambda} = \frac{n(t_s)}{(1 - e^{-\mu t_s})} . \quad (3.5)$$

$\hat{\lambda}$ is substituted into the partial derivative of l with respect to μ to give,

$$\partial l / \partial \mu = -n(t_s) \frac{t_s e^{-\mu t_s}}{1 - e^{-\mu t_s}} - \Delta \bar{n}(t_s) + \frac{n(t_s) \Delta e^{-\mu \Delta}}{1 - e^{-\mu \Delta}} = 0 , \quad (3.6)$$

where

$$\bar{n}(t_s) = \sum_{j=1}^J (j-1) n_j . \quad (3.7)$$

μ can now be solved for from the following equation:

$$\frac{\Delta e^{-\mu\Delta}}{1-e^{-\mu\Delta}} - \frac{t_s e^{-\mu t_s}}{1-e^{-\mu t_s}} = \frac{\Delta \bar{n}(t_s)}{n(t_s)} . \quad (3.8)$$

This equation closely resembles Schneidewind's result; see (2.41). Since $t_s = \Delta J$, equation (3.8) becomes

$$\frac{e^{-\mu\Delta}}{1-e^{-\mu\Delta}} - \frac{J e^{-\mu t_s}}{1-e^{-\mu t_s}} = \frac{\bar{n}(t_s)}{n(t_s)} , \quad (3.9)$$

then,

$$\frac{e^{-\mu\Delta}}{1-e^{-\mu\Delta}} = J \frac{(e^{-\mu\Delta})^J}{1-(e^{-\mu\Delta})^J} + \frac{\bar{n}(t_s)}{n(t_s)} = r . \quad (3.10)$$

By letting $x = e^{-\mu\Delta}$ into equation (3.10) becomes,

$$\frac{x}{1-x} = J \frac{x^J}{1-x^J} + \frac{\bar{n}(t_s)}{n(t_s)} = r ; \quad (3.11)$$

x is solved for iteratively. Let $J=0$ for the first iteration, then

$$r(1) = \frac{x(1)}{1-x(1)} = \frac{\bar{n}(t_s)}{n(t_s)} , \quad (3.12)$$

$$x(1) = \frac{\bar{n}(t_s)}{\bar{n}(t_s) + n(t_s)} = \frac{r(1)}{1+r(1)} . \quad (3.13)$$

$r(2)$ is

$$r(2) = r(1) + J \frac{x(1)^J}{1-x(1)^J} . \quad (3.14)$$

$x(2)$ is given by,

$$x(2) = \frac{r(2)}{1+r(2)} . \quad (3.15)$$

Hence the iteration of $r(n)$ and $x(n)$ is

$$r(n+1) = r(n) + J \frac{X(n)^J}{1-X(n)^J} , \quad (3.16)$$

$$x(n+1) = \frac{r(n+1)}{1+r(n+1)} . \quad (3.17)$$

The iterative process continues until $x(n+1) - x(n) < \epsilon$, where ϵ is a suitable small number; $x(n+1)$ is then substituted into equation (3.5) to get $\hat{\lambda}$. Using the estimates of $\hat{\mu}$ and $\hat{\lambda}$, the expected number of faults to be observed in some additional operating time t_0 , where $(t_s, t_s + t_0)$ is of length $k\Delta$, can be estimated

$$\hat{E}[N(t_0) - N(t_s)] = n(t_s) \left\{ \frac{e^{-\hat{\mu} t_s}}{1 - e^{-\hat{\mu} t_s}} (1 - e^{-\hat{\mu} k\Delta}) \right\} . \quad (3.19)$$

A Bayesian methodology is discussed in the appendix. This method attempts to utilize past experience from software projects having similar characteristics as the software in question. If the distributions of λ and μ are known from experience then this information can be useful in estimating the parameters $\hat{\lambda}$ and $\hat{\mu}$.

B. BOOTSTRAP

Bootstrapping was used to obtain the confidence limits for $\hat{\mu}$, $\hat{\lambda}$, and $\hat{E}[N(t_0) - N(t_s)] = \hat{E}[\Delta N(t_0)]$. This technique takes into account the sampling uncertainties in the estimates by removing the errors in the standard approximation (Dalal et al, 1989) and (Efron, 1985). To obtain the estimates of the sampling variability of $\hat{\mu}$, $\hat{\lambda}$, and $\hat{E}[N(t_0) - N(t_s)] = \hat{E}[\Delta(t_0)]$ proceed as follows. The probability that a count occurs in the j th period is conditional on $N(t_s) = n(t_s)$:

$$P\{N_1 = n_1, \dots, N_J = n_J | N_1 + N_2 + \dots + N_J = n(t_s)\} \quad (3.20a)$$

$$= \prod_{j=1}^J \frac{n(t_s)!}{n_j!} \left(\frac{\lambda_j}{\sum \lambda_i} \right)^{n_j}, \quad (3.20b)$$

where $\sum \lambda_i = 1 - e^{-j\mu\Delta}$. From this the probability that a count falls in the j th interval is

$$P_j = \frac{1 - e^{-j\mu\Delta}}{1 - e^{-J\mu\Delta}}. \quad (3.21)$$

Uniform (0,1) random numbers were generated, where the $k=1, 2, \dots, n(t_s)$; U_k is the k th random number. If $P_{(j-1)} < U_k \leq P_j$ then a count is added to n_j . The simulated n_j 's were then used to re-estimate $\hat{\mu}$, $\hat{\lambda}$, and $\hat{E}[\Delta N(t_0)]$; these are the bootstrap values. This process was repeated 1000 times to get a range of values for $\hat{\mu}$, $\hat{\lambda}$, and $\hat{E}[\Delta N(t_0)]$. To create a 90% confidence limit of the estimate $\hat{E}[\Delta N(t_0)]$ the 1000 bootstrap estimates

of $\hat{E}[\Delta N(t_0)]$ were ordered and the values of the 50th and 950th quantiles were found. These are quoted as the 90% confidence region $(\hat{E}[\Delta N(t_0)]_{.5}, \hat{E}[\Delta N(t_0)]_{.95})$.

C. RESULTS

The estimates for the parameters were obtained using three different Δ values and three different t_i values. The value t_0 was selected such that $t_0 + t_i$ = time of last observed fault to occur; this allows for comparison of the predicted expected number of faults to occur with the observed data. The data provided in Tables 1 through 6 are the 90% confidence interval obtained by the bootstrap. The most difficult aspect of this thesis research was obtaining appropriate test data. The data that I received from various sources was unacceptable for various reasons: no testing history, severity of faults not listed, no milestone events listed (i.e. one data set covered 10 years but no indication of modifications to the software), non-software errors listed with software errors, description of errors could not be interpreted (which may have eliminated some of the problems mentioned above). The underlying cause of this is that organizations that I contacted for data do not use any systematic method for determining software reliability. A "warm fuzzy feeling" for the software seems to be the current method used to judge the reliability of the software. This feeling gets warmer and fuzzier as deadlines draw closer. The data sets used in the analysis of the model

were obtained from a technical report on other software reliability models (Abdalla et. al., 1986). The data was given as time (CPU) between failures. The results of the bootstrap for Data Set 1 are given in Tables 1-3; the graphical results (Dalal, 1990) are depicted in Figures 1-3. The results of the bootstrap for Data Set 2 are given in Tables 4-6; the graphical results (Dalal, 1990) are depicted in Figures 4-5.

D. USE OF RESULTS

Suppose a time t_1 has been spent testing the software, and $n(t_1)$ faults were found. The $n(t_1)$ faults can be broken up into n_j 's, the number of faults in each period j of size Δ ($\sum n_j = n(t_1)$). This information can be used to estimate the parameters $\hat{\mu}$ and $\hat{\lambda}$, and a point estimate of the mean or expected number of faults to appear in the time interval (t_1, t_1+t_0) . Operational testing of the system will require some time t_0 . Bootstrapping can now be done to assess the sampling uncertainty in the estimate of the expected number of faults to appear in (t_1, t_1+t_0) . This will be done by quoting bootstrapped 90% confidence limits. The expected number of faults predicted to occur can be compared to the requirements of the system i.e. for some time t_0 for example; at most F faults are allowed (suppose F can be specified). If the predicted expected number of faults is less than the allowable number of faults then system operational testing might be

worth the expense at this time. In contrast to this, if the expected number of faults is greater than the specified number of faults then system operational testing should be postponed. Testing should continue in the lab, at the developmental level until t_0 and $n(t_0)$ are large enough that the expected number of faults for the required operational time meets specification.

A more conservative approach is to replace the estimate of the mean number of faults by the upper confidence limit of the mean number of faults. Such a conservative approach is recommended.

If there are no specifications the individual responsible for scheduling system operational testing will have to make a subjective decision. Is the expected number of faults to occur in (t_0, t_0+t_1) small enough to warrant spending the money to carry out system operational testing, or should this testing be postponed until the expected number of faults is lower. The assumption is that lab testing will continue on the software, increasing t_0 and $n(t_0)$, but reducing the number of unfound and uncorrected faults. The more faults found in lab testing of the software the fewer the number of faults that are likely to occur in the more costly system operational testing.

E. APPLICATION TO TWO DATA SETS

The fitting and error assessment procedure was applied to two data sets (Abdalla et al, 1986). Figures 1, 2, and 3 refer to Data Set 1; Figures 4, 5, and 6 to Data Set 2.

Figure 1 has a Δ of 10 CPU minutes with three combinations of t_i and t_o . If the range of the expected number of faults for $t_i=1250$, $t_o=250$ (2.21 to 6.09) is acceptable the software manager may choose to schedule operational testing. The same argument can be made for $t_i=1000$, $t_o=500$. A problem occurs for $t_i=500$ and $t_o=1000$. If the range for the expected number of faults to occur (4.69 to 22.22) is acceptable the software manager may choose to schedule operational testing. Unfortunately, 46 faults occur in (t_i, t_i+t_o) . This is extremely likely to be the result of use of an inappropriate model (it does seem unlikely that software with as many as 22 mission-critical faults would be viewed as acceptable for starting operational testing). What can the software manager do to prevent something like this from occurring? Ideally, as testing continues, the rate at which faults occur should decrease (assuming a constant relative rate of testing), with that rate asymptotically approaching zero as t_i becomes large. The slope of the estimated total expected number of faults verses test time for Data Set 1 from $T=300$ to $T=500$ is $m=0.08$ (faults/cpu min). Figure 1 depicts this: the rate at which faults are occurring does not appear to be tapering off. The

software manager can use this information to support a decision to go ahead with (or postpone) operational testing. From $T=1000$ to $T=1500$ the slope is 0.028 (faults/cpu min) and appears to be tapering off. The range of the expected number of faults to occur in the specified t_0 accurately reflect what actually occurred. If the range of the expected number of faults is acceptable the software manager should go ahead with operational testing. Figure 2 ($\Delta = 20$ cpu minutes) and Figure 3 ($\Delta = 50$ cpu minutes) can be interpreted similarly.

The change in Δ for both data sets did not have a significant impact on the range of the expected number of faults to occur, indicating that the model is somewhat insensitive to the size of Δ .

Data Set 2 (Figures 4,5, and 6) shows only a small indication of the slope decreasing. This is why the confidence limits of the expected number of faults is so wide. The software manager can apply the same techniques listed above to make a decision to schedule (or postpone) operational testing. The software manager must repeatedly address the questions: is the rate of occurrence of faults lessening, and is the range of expected number of faults acceptable to support operational testing?

A fitted model may indicate a narrowing range of expected number of faults and slope asymptotically approaching zero, consequently the software manager schedules operational

testing. Unfortunately, the results of the operational testing may be poor i.e. a relatively large number of errors may occur indicating that more developmental activity and testing is required to improve the software. For example, the model predicts $n(t_0) \approx 22$ for Data Set 1 ($t_s=500$, $t_0=1000$), but the number of observed faults that occurred in t_0 was more than twice the predicted amount, 46. This example illustrates the relationship between modeling and testing. While a systematic underestimation indicates flaws in the model, occasional underestimation simply reinforce that software reliability models do not take the place of stressing software within a full system in a real-life operational environment. The purpose of this thesis is to provide the software manager with a tool to aid in the decision as to when to initiate operational testing, not to replace such a test.

TABLE 1 ESTIMATE OF PARAMETERS FOR DATA SET 1 $t_1=1250$, $t_0=250$ (CPU MINUTES) Observed number of bugs in t_0 is 6 90% Confidence Interval			
Δ (CPU min)	$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10 5 %	0.00272	134.595	2.21
95 %	0.00176	146.509	5.76
20 5 %	0.00270	134.993	2.32
95 %	0.00174	147.798	6.09
50 5 %	0.00270	135.258	2.25
95 %	0.00175	148.142	5.82

TABLE 2 ESTIMATE OF PARAMETERS FOR DATA SET 1 $t_1=1000$, $t_0=500$ (CPU MINUTES) Observed number of bugs in t_0 is 14 90% Confidence Interval			
Δ (CPU min)	$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10 5 %	0.00298	128.701	5.03
95 %	0.00177	147.640	14.73
20 5 %	0.00298	128.969	5.07
95 %	0.00176	148.393	14.81
50 5 %	0.00296	129.828	5.17
95 %	0.00175	150.549	14.96

TABLE 3 ESTIMATE OF PARAMETERS FOR DATA SET 1 $t_1=500$, $t_0=1000$ (CPU MINUTES) Observed number of bugs in t_0 is 46 90% Confidence Interval			
Δ (CPU min)	$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10 5 %	0.00600	95.010	4.69
95 %	0.00327	112.711	20.97
20 5 %	0.00600	95.352	4.70
95 %	0.00326	113.863	21.14
50 5 %	0.00588	96.859	5.00
95 %	0.00317	118.432	22.22

TABLE 4
ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_s=800$, $t_o=300$ (CPU SECONDS)
Observed number of bugs in t_o is 12
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_o)]$
10	5 %	0.00288	82.479	4.75
	95 %	0.00111	126.998	14.69
20	5 %	0.00288	82.718	4.73
	95 %	0.00111	127.645	14.64
50	5 %	0.00287	83.722	4.78
	95 %	0.00111	131.003	14.66

TABLE 5
ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_s=600$, $t_o=500$ (CPU SECONDS)
Observed number of bugs in t_o is 21
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_o)]$
10	5 %	0.00298	78.513	10.10
	95 %	0.00068	195.611	37.09
20	5 %	0.00296	79.189	10.21
	95 %	0.00067	200.950	37.32
50	5 %	0.00298	80.710	10.14
	95 %	0.00067	211.307	37.43

TABLE 6
ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_s=400$, $t_o=700$ (CPU SECONDS)
Observed number of bugs in t_o is 37
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_o)]$
10	5 %	0.00456	58.950	9.03
	95 %	0.00058	239.964	62.43
20	5 %	0.00458	59.423	8.96
	95 %	0.00054	263.011	63.88
50	5 %	0.00446	62.014	9.45
	95 %	0.00047	325.387	66.55

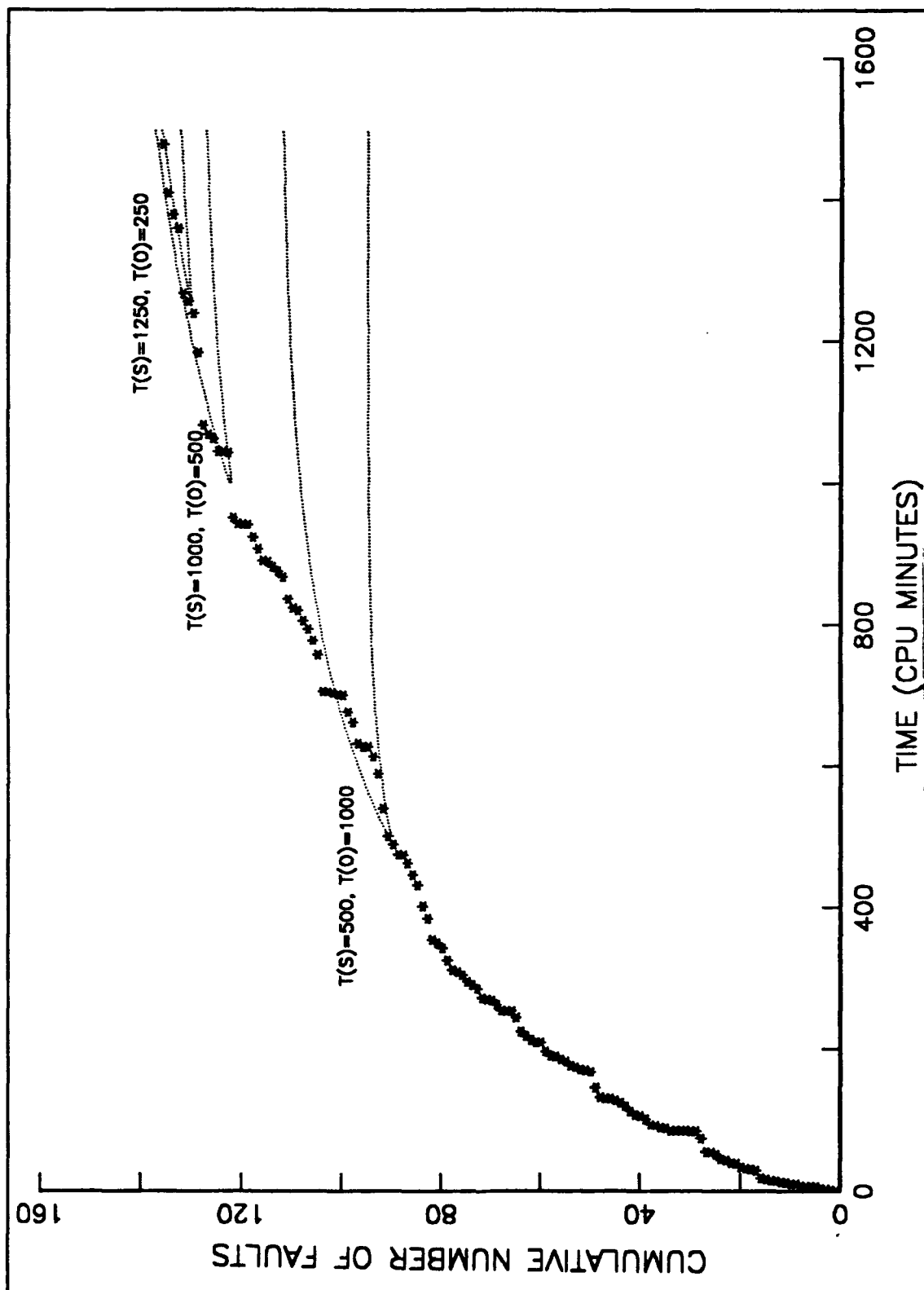


Figure 1. Data Set 1, $\Delta = 10$ (CPU minutes)

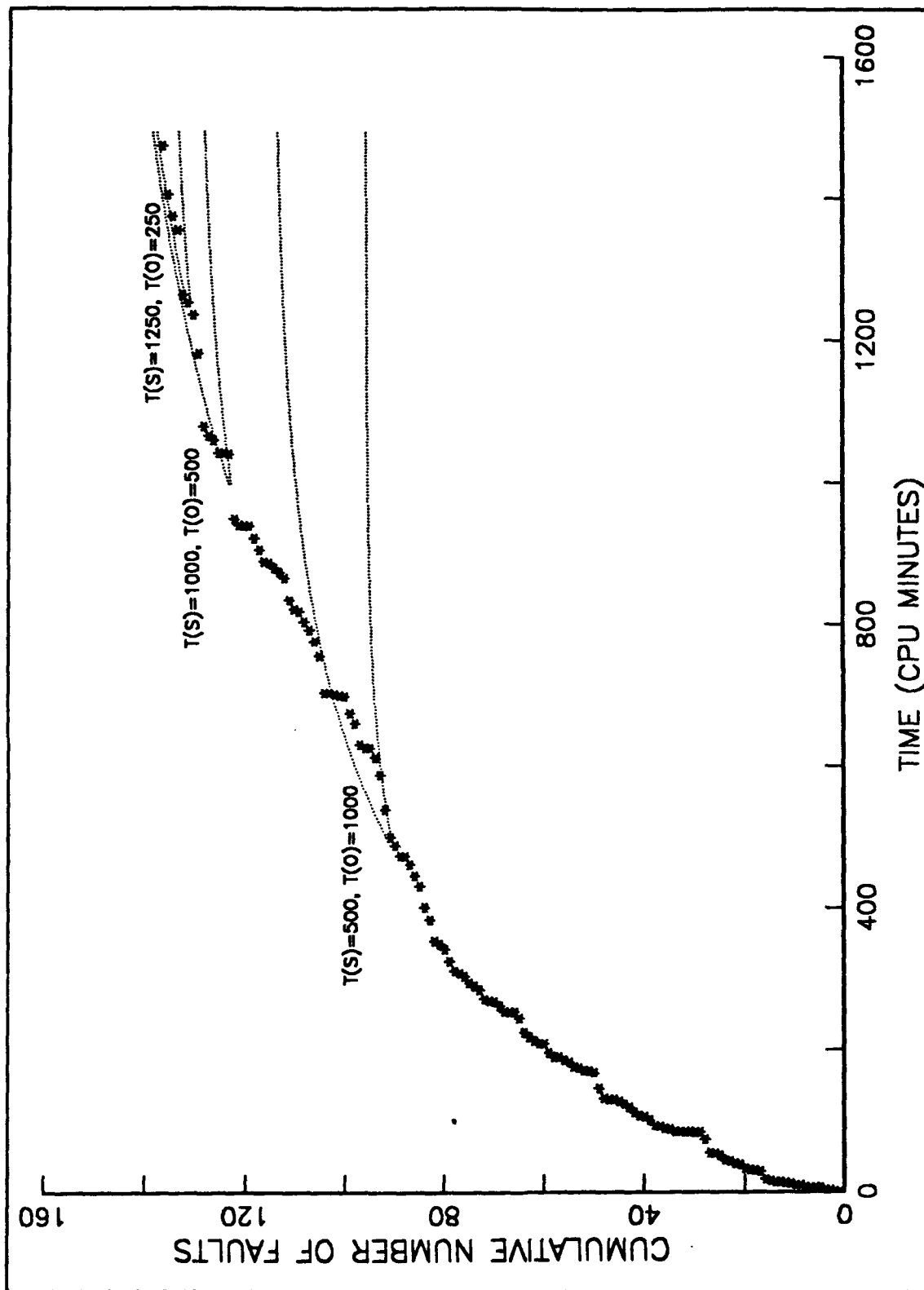


Figure 2. Data Set 1, $\Delta = 20$ (CPU minutes)

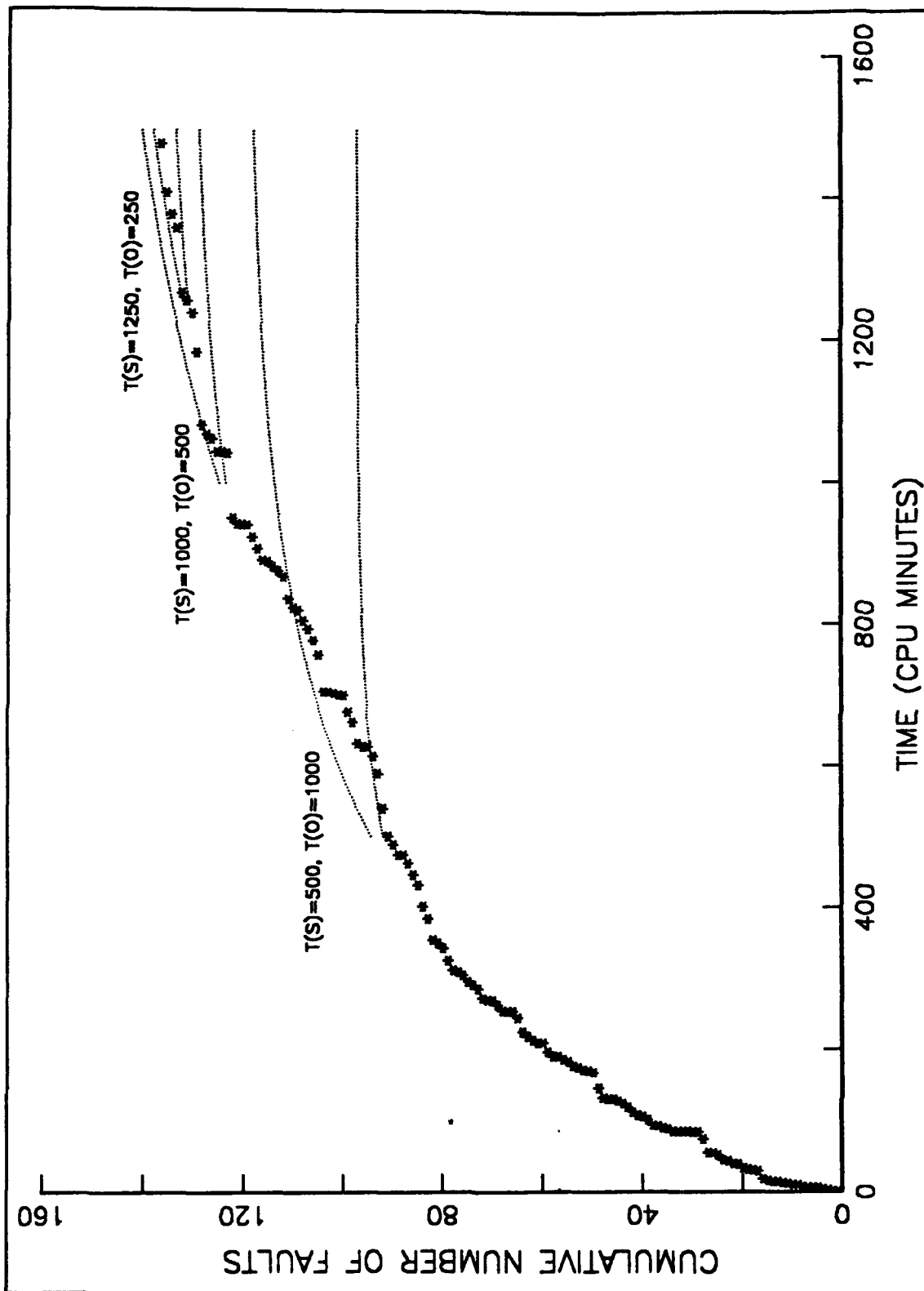


Figure 3. Data Set 1, $\Delta = 50$ (CPU minutes)

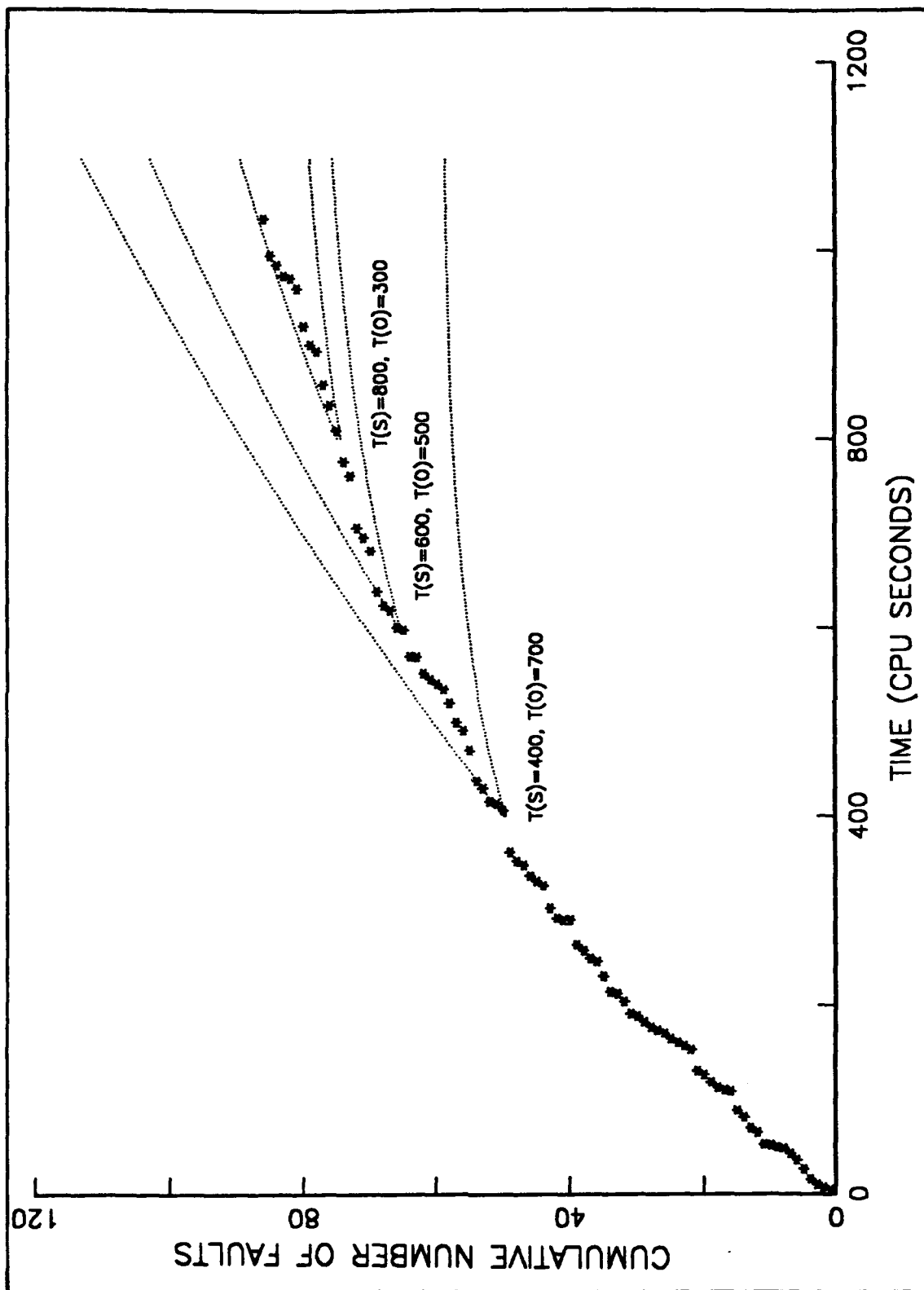


Figure 4. Data Set 2, $\Delta = 10$ (CPU seconds)

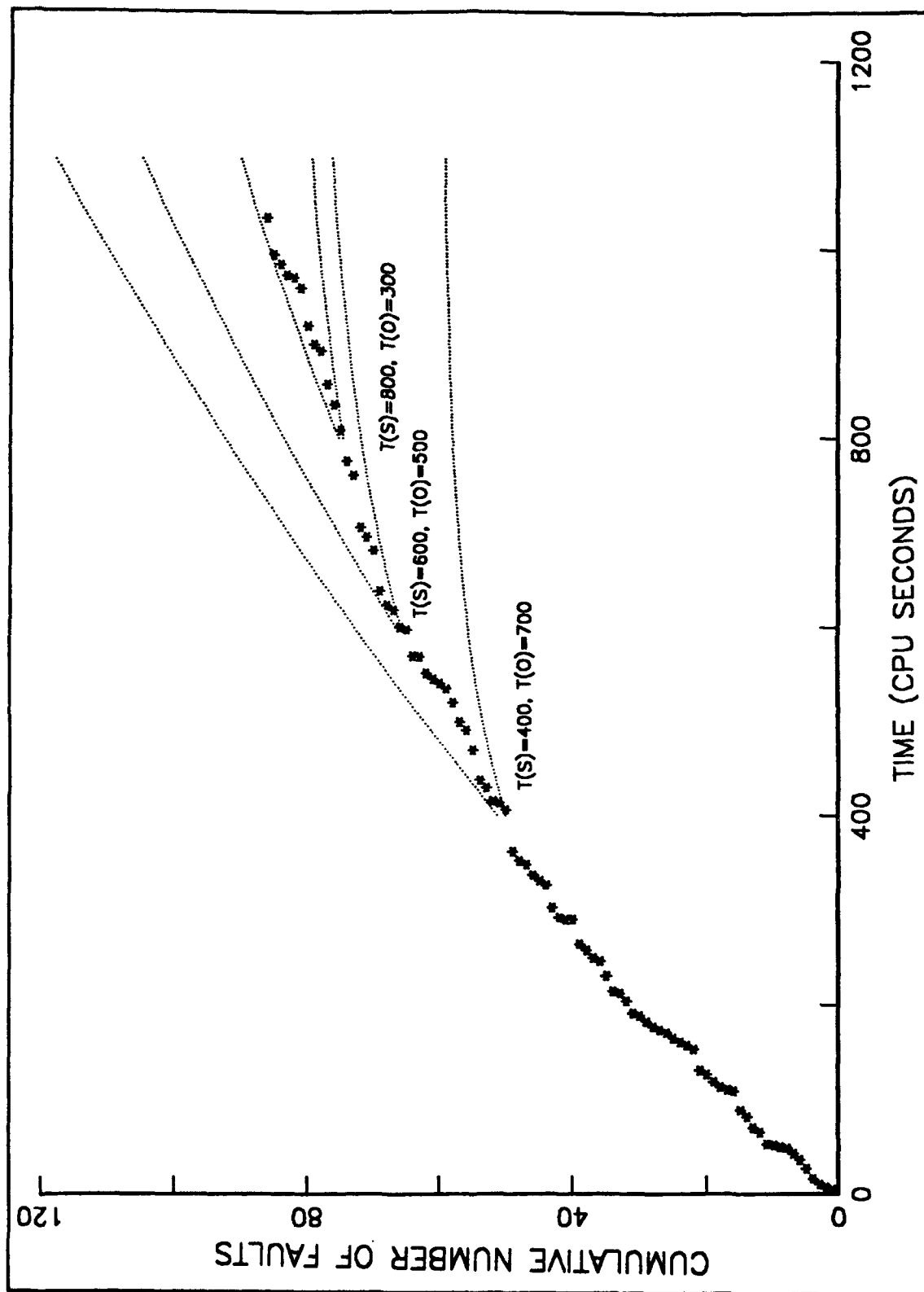


Figure 5. Data Set 2, $\Delta = 20$ (CPU seconds)

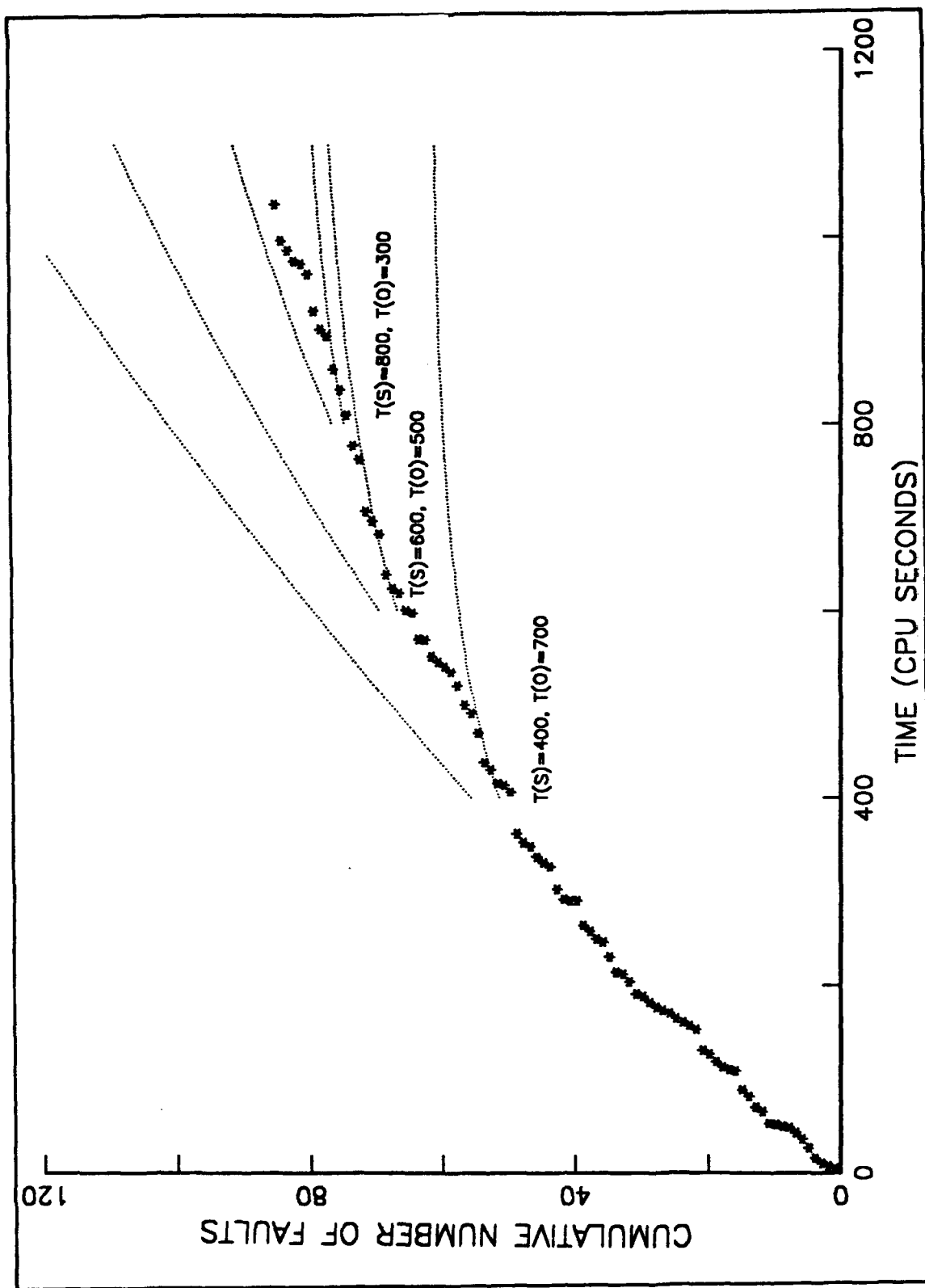


Figure 6. Data Set 2, $\Delta = 50$ (CPU seconds)

IV. CONCLUSION

Software reliability models are useful tools that managers of software intensive projects have at their disposal. The bootstrapping technique will provide the manager a range of expected number of faults estimated to occur for some additional operating time. The question is, is the upper limit of the expected number of faults estimated to occur acceptable? The potential risks are additional cost for further testing or late product delivery. The ideal case is reliable software delivered on time and on budget. Unfortunately, reality is rarely ideal. The software manager must decide: is it better to deliver a product on time that may be considered unreliable by the user and be sent back for further testing, or to deliver a product late but of acceptable quality to the user? The purpose of this thesis is to provide a quantitative tool for the manager who may have to make such qualitative decisions. The use of software reliability models is not without associated cost, and risk. The data must be collected for input to the model. Recommendations for the type of data that should be collected are:

- Operating time between failures (CPU time is the best) (Musa and Okumoto, 1984).
- Calendar time between failures, although such times may not accurately reflect the opportunity for faults to reveal themselves (Musa et al, 1987).

- Testing history i.e. how many people are involved in the testing effort.
- How the software was tested
- Intensity of the software testing
- Cost of testing i.e. the cost to find and repair a fault before and after product delivery.

Without useful data a reliability model has little practical use. The model presented in this thesis should be validated using data from several Navy systems.

There are several areas for further research. How accurate are the predicted confidence limits in this model? What are the limits of applicability of this model? What effect do inaccuracies (due to replacing observed data with hypothesized data in cases where insufficient data is available) have on the model i.e. how robust is the model? Further development of other software reliability models should be pursued. Emphasis should be placed on obtaining confidence limits in addition to quoting only a point estimate of the expected number of failures predicted to appear for some additional testing time. These models should be verified using data obtained from Navy software intensive systems. It is infeasible to test every possible branch in a large program for faults. The software manager needs technical assistance in identifying where effort and money should be spent to deliver the best possible product. Will many faults in portions of the software that are rarely used/reached cause

more problems for the user than a few faults in frequently used/reached portions.

APPENDIX

Software projects may have similar characteristics such as testing strategies or architecture, so that the information obtained about the reliability of one software project may be used to aid in the prediction of the reliability of another similar, software project. This process can make use of Bayesian methodology (Dalal and Mallows, 1990), (Farr, 1983). If prior distributions of λ and μ are specified then this information can be used help estimate the parameters λ and μ ; the posterior for these is

$$p_{\lambda, \mu}(\lambda, \mu) = KL(\lambda, \mu) p_{\lambda}(\lambda) p_{\mu}(\mu) , \quad (a.1a)$$

$$= K e^{-\lambda(1-e^{-\mu t_s})} \lambda^{n(t_s)} \left(\prod_{j=1}^J e^{-\mu \Delta(j-1)n_j} (1-e^{-\mu \Delta})^{n(t_s)} \right) p_{\lambda}(\lambda) p_{\mu}(\mu) , \quad (a.1b)$$

where $p_{\lambda}(\lambda)$ and $p_{\mu}(\mu)$ are the prior distributions of λ and μ estimated from another software project that has characteristics similar to the software project currently being tested. The simplest idea is to integrate out λ and marginalize on μ which yields:

$$p_{\mu}(\mu) = K \int_0^{\infty} e^{-\lambda(1-e^{-\mu t_s})} \lambda^{n(t_s)} p_{\lambda}(\lambda) d\lambda \cdot e^{-\mu \Delta \bar{n}(t_s)} (1-e^{-\mu \Delta})^{n(t_s)} . \quad (a.2)$$

The most convenient choice of $p_\lambda(\lambda)$ is the (conjugate) Gamma:

$$p_\lambda(\lambda) = e^{-\alpha\lambda} \frac{(\alpha\lambda)^{\beta-1}}{\Gamma(\beta)} , \quad (a.3)$$

which when substituted into equation (a.2) yields the density,

$$p_\mu(\mu) = K e^{-\mu\Delta\bar{n}(t_s)} (1 - e^{-\mu\Delta})^{n(t_s)} \int_0^\infty e^{-\lambda(1-e^{-\mu\Delta})} \lambda^{n(t_s)} e^{-\alpha\lambda} \frac{(\alpha\lambda)^{\beta-1}}{\Gamma(\beta)} , \quad (a.4a)$$

$$= K' e^{-\mu\Delta\bar{n}(t_s)} (1 - e^{-\mu\Delta})^{n(t_s)} \int_0^\infty \frac{e^z z^{n(t_s)+\beta-1} dz}{(\alpha+1-e^{-\mu\Delta})^{n(t_s)+\beta}} , \quad (a.4b)$$

$$= K'' e^{-\mu\Delta\bar{n}(t_s)} (1 - e^{-\mu\Delta})^{n(t_s)} \frac{1}{(\alpha+1-e^{-\mu\Delta})^{n(t_s)+\beta}} . \quad (a.4c)$$

Using an uninformative prior, $\alpha=0$, $\beta=0$, and setting $x=e^{-\mu\Delta}$ equation (a.4c) becomes

$$p_x(x) = K^* x^{\bar{n}(t_s)} (1-x)^{n(t_s)} \frac{1}{(\alpha+1-x)^{n(t_s)+\beta}} . \quad (a.5)$$

The mode of the density is

$$I(x) = \ln(p_x(x)) = \bar{n}(t_s) \ln x + n(t_s) \ln(1-x) - (n(t_s) + \beta) \ln(\alpha+1-x) . \quad (a.6)$$

Taking the partial derivative of equation (a.7) with respect to x yields:

$$\frac{\bar{n}(t_s)}{n(t_s)} = \frac{x}{1-x} - \frac{n(t_s) + \beta}{n(t_s)} \times \frac{Jx^J}{\alpha + 1 - x^J} . \quad (a.7)$$

If $\alpha = \beta = 0$ equation (a.7) is the same as equation (3.11), which gives the MLE.

Suppose $m = E[\lambda]$ and $\sigma^2 = \text{Var} [\lambda]$ in the prior, then $\alpha = m/\sigma^2$ and $\beta = m(m/\sigma^2)$. Equation (a.7) is

$$\frac{x}{1-x} - \frac{n(t_s) + m(m/\sigma^2)}{n(t_s)} \times \frac{Jx^J}{(m/\sigma^2) + 1 - x^J} = \frac{\bar{n}(t_s)}{n(t_s)} . \quad (a.8)$$

If λ is interpreted as the total number of faults in a particular software project, then the number of faults is discrete so a discrete distribution should be used for the prior, i.e. one could use a Poisson for the prior. However, it is easier to work with a Gamma distribution. If the Gamma distribution has same parameters as a Poisson then equation (a.8) is (since $m = \sigma^2$)

$$\frac{x}{1-x} - \frac{n(t_s) + m}{n(t_s)} \times \frac{Jx^J}{2 - x^J} = \frac{\bar{n}(t_s)}{n(t_s)} . \quad (a.9)$$

It is clear that the variance to mean ratio of the prior has strong influence on the effect of a prior estimate of the mean.

One Bayesian approach to estimation is to find the mean (rather than the mode, or highest point of the posterior as is essentially done in the likelihood approach) of the

(approximate) posterior, $0 \leq x \leq 1$. To obtain an approximate posterior mode proceed as follows. If J is large x' is small provided $x > 0$, so expand in Taylor's series to get

$$p_{\frac{1}{2}}(x) = k^{**} [x^{\bar{n}}(1-x)^n + (\frac{n+\beta}{1+\alpha}) (x^{\bar{n}+J}(1-x)^n)] , \quad (a.10)$$

where: $n=n(t_s)$ and $\bar{n}=\bar{n}(t_s)$.

Equation (a.10) is a convex combination of two beta densities. K^{**} can be found by setting the left hand side of (a.11) = 1. $E[\underline{x}] = E[e^{-\mu\Delta}]$ can be found,

$$\underline{x} = k^{**} \left[\frac{\Gamma(\bar{n}+1)\Gamma(n+1)}{\Gamma(\bar{n}+n+1)} \frac{\bar{n}+1}{\bar{n}+n+1} \right. \quad (a.11)$$

$$\left. + (\frac{n+\beta}{1+\alpha}) \frac{\Gamma(\bar{n}+J+1)\Gamma(n+1)}{\Gamma(\bar{n}+J+n+1)} \frac{\bar{n}+J+1}{\bar{n}+J+n+1} \right] .$$

The approximation to this is

$$\hat{x} = \frac{\frac{\bar{n}!}{(\bar{n}+n)!} \frac{\bar{n}+1}{\bar{n}+n+1} + \frac{n+\beta}{1+\alpha} \frac{(\bar{n}+J)!}{(\bar{n}+J+n)!} \frac{\bar{n}+J+1}{\bar{n}+J+n+1}}{\frac{\bar{n}!}{(\bar{n}+n)!} + \frac{n+\beta}{1+\alpha} \frac{(\bar{n}+J)!}{(\bar{n}+J+n)!}} = e^{\mu\Delta} . \quad (a.12)$$

Unfortunately, $n=n(t_s)=136$ for Data Set 1; even with factoring out $\bar{n}=\bar{n}(t_s)$, the factorial ratios are on the order of 10^{-300} . However, it is justifiable to use an approximation to the factorials to get

$$\hat{x} = \frac{\frac{\bar{n}+1}{\bar{n}+n+1} + \frac{n+\beta}{1+\alpha} \frac{\bar{n}+J+1}{\bar{n}+n+J+1} (\frac{\bar{n}+1}{\bar{n}+n+1})^J}{1 + \frac{n+\beta}{1+\alpha} (\frac{\bar{n}+1}{\bar{n}+n+1})^J} . \quad (a.13)$$

The numerical results of equation (a.13) are in Tables A1 through A6 for Data Sets 1 and 2. The graphical results are shown in Figures A1 through A6. The range of the estimated number of faults to occur in (t_i, t_i+t_0) is much smaller than that of the bootstrap results discussed in Chapter III. None of the results (estimated number of faults to occur) using the Bayesian method contain the observed faults. A possible explanation for this is inappropriate values for α and β ($\alpha=\beta=0$). After various projects have been analyzed with software reliability models, fault distribution may become more apparent. This information can then be incorporated to reliability models. I feel that, despite the surprising initial results, this method does promise to be a useful tool to the software manager.

TABLE A1				
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 1				
$t_1=1250, t_0=250$ (CPU MINUTES)				
Observed number of bugs in t_0 is 6				
90% Confidence Interval				
Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00339	131.892	1.08
	95 %	0.00263	135.028	2.42
20	5 %	0.00340	131.870	1.10
	95 %	0.00264	134.992	2.48
50	5 %	0.00339	131.914	1.09
	95 %	0.00262	135.103	2.45

TABLE A2				
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 1				
$t_1=1000, t_0=500$ (CPU MINUTES)				
Observed number of bugs in t_0 is 14				
90% Confidence Interval				
Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00399	124.289	1.98
	95 %	0.00311	127.719	4.51
20	5 %	0.00399	124.304	1.99
	95 %	0.00310	127.768	4.54
50	5 %	0.00398	124.328	2.01
	95 %	0.00309	127.828	4.58

TABLE A3				
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 1				
$t_1=500, t_0=1000$ (CPU MINUTES)				
Observed number of bugs in t_0 is 46				
90% Confidence Interval				
Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00808	91.608	1.61
	95 %	0.00602	94.660	4.65
20	5 %	0.00809	91.603	1.60
	95 %	0.00601	94.697	4.69
50	5 %	0.00797	91.708	1.71
	95 %	0.00596	94.805	4.79

TABLE A4
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_1=800$, $t_0=300$ (CPU SECONDS)
Observed number of bugs in t_0 is 12
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00464	75.849	1.39
	95 %	0.00340	79.192	3.32
20	5 %	0.00464	75.846	1.39
	95 %	0.00340	79.216	3.34
50	5 %	0.00465	75.837	1.38
	95 %	0.00339	79.242	3.35

TABLE A5
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_1=600$, $t_0=500$ (CPU SECONDS)
Observed number of bugs in t_0 is 21
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00600	66.830	1.74
	95 %	0.00429	70.363	4.74
20	5 %	0.00600	66.828	1.74
	95 %	0.00429	70.361	4.73
50	5 %	0.00596	66.872	1.78
	95 %	0.00429	70.368	4.74

TABLE A6
BAYESIAN ESTIMATE OF PARAMETERS FOR DATA SET 2
 $t_1=400$, $t_0=700$ (CPU SECONDS)
Observed number of bugs in t_0 is 37
90% Confidence Interval

Δ (CPU min)		$\hat{\mu}$	$\hat{\lambda}$	$\hat{E}[N(t_0)]$
10	5 %	0.00897	50.391	1.39
	95 %	0.00625	53.386	4.33
20	5 %	0.00893	50.416	1.41
	95 %	0.00624	53.397	4.34
50	5 %	0.00891	50.426	1.42
	95 %	0.00622	53.432	4.38

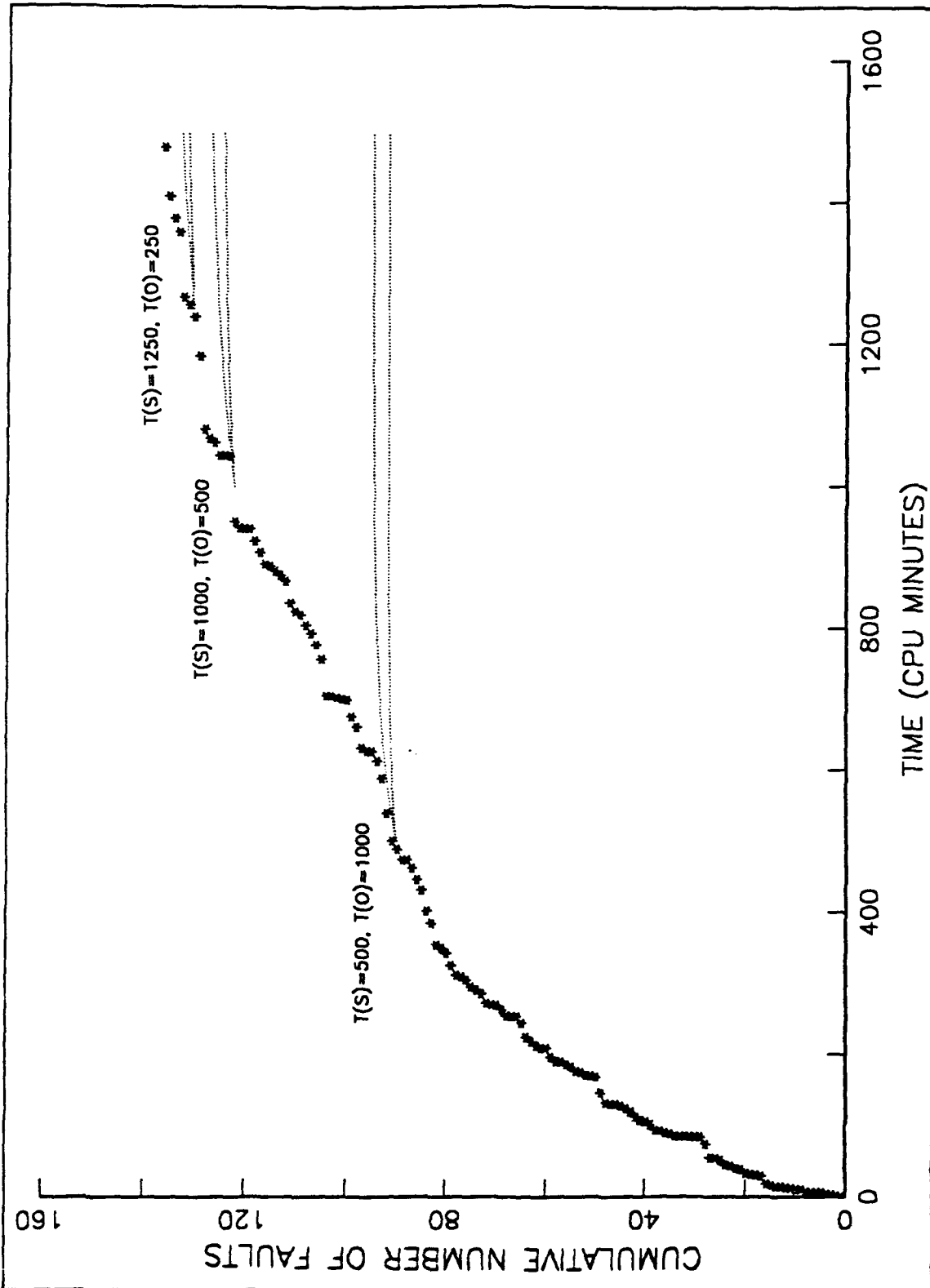


Figure A1. Data Set 1, $\Delta = 10$ (CPU minutes), Bayesian Method

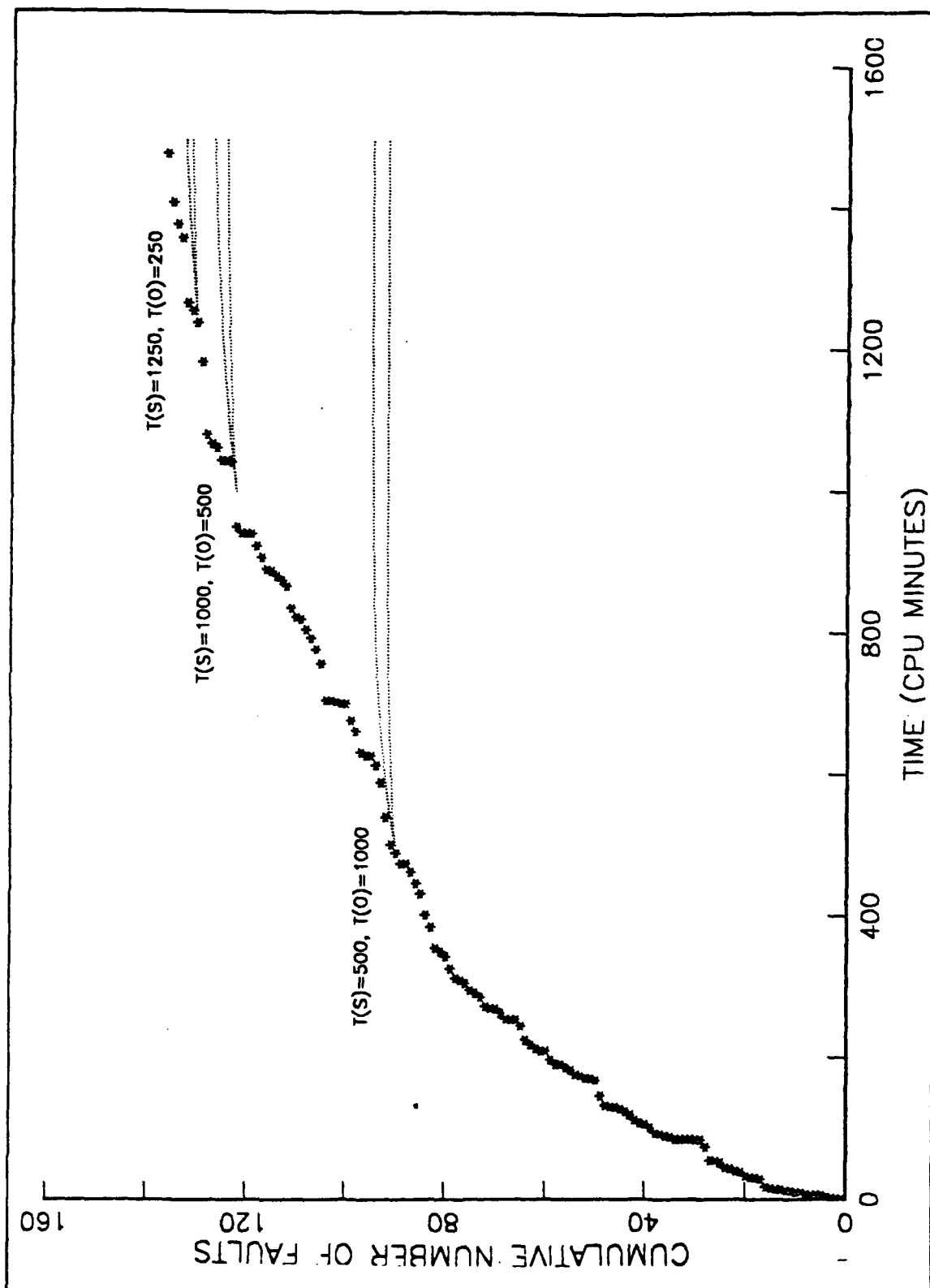


Figure A2. Data Set 1, $\Delta = 20$ (CPU minutes), Bayesian Method

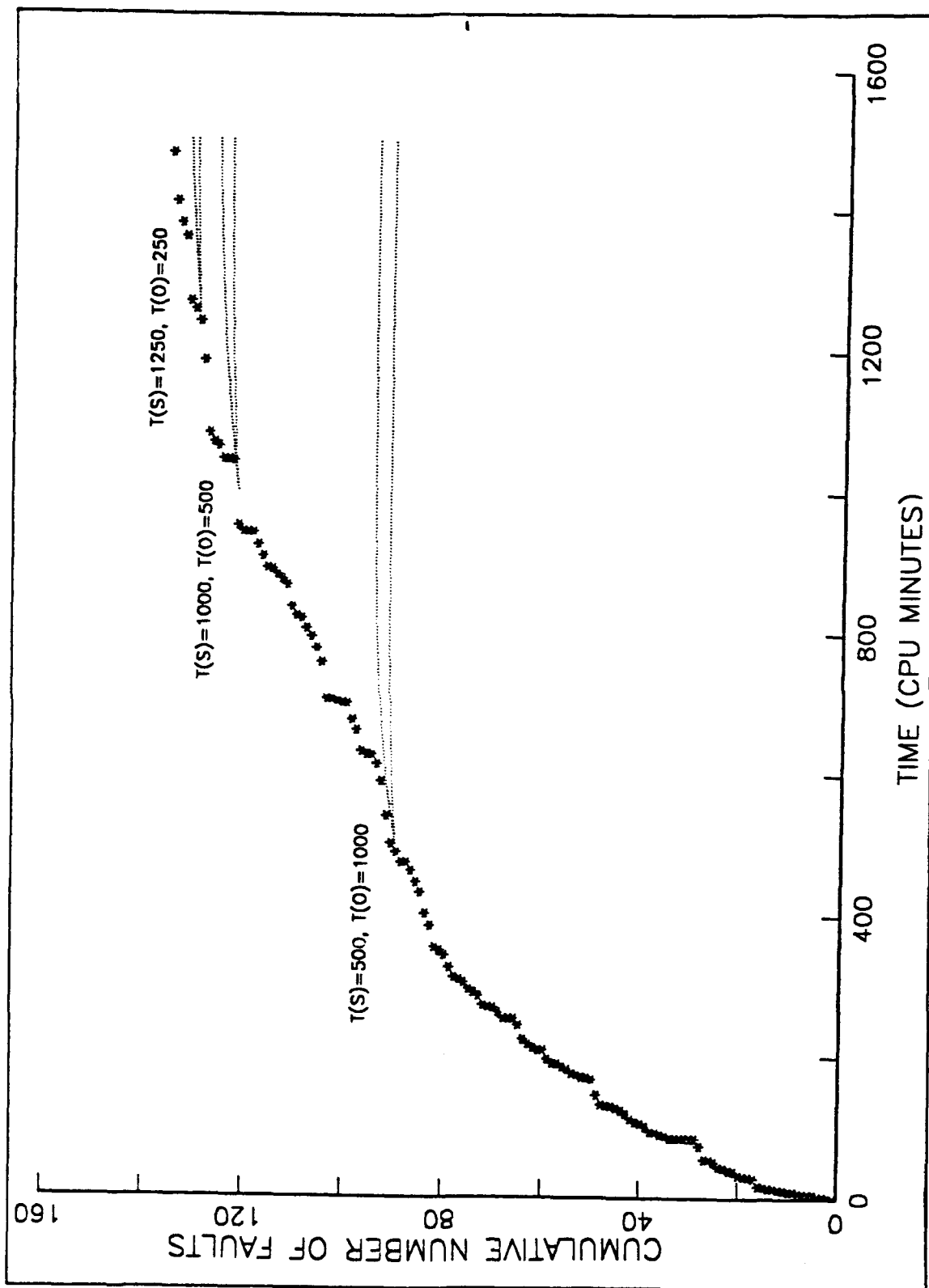


Figure A3. Data Set 1, $\Delta = 50$ (CPU minutes), Bayesian Method

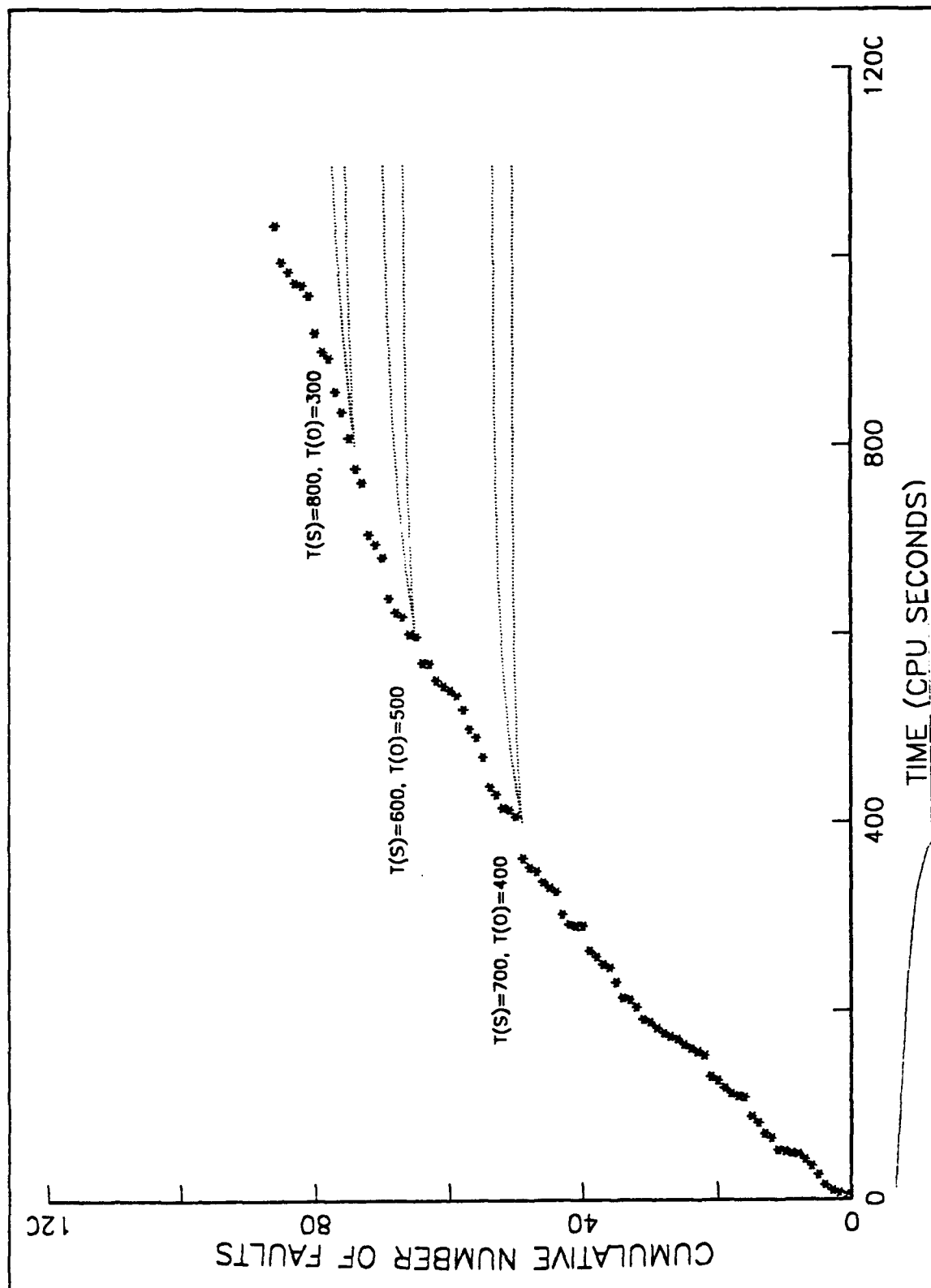


Figure A4. Data Set 2, $\Delta = 10$ (CPU seconds), Bayesian Method

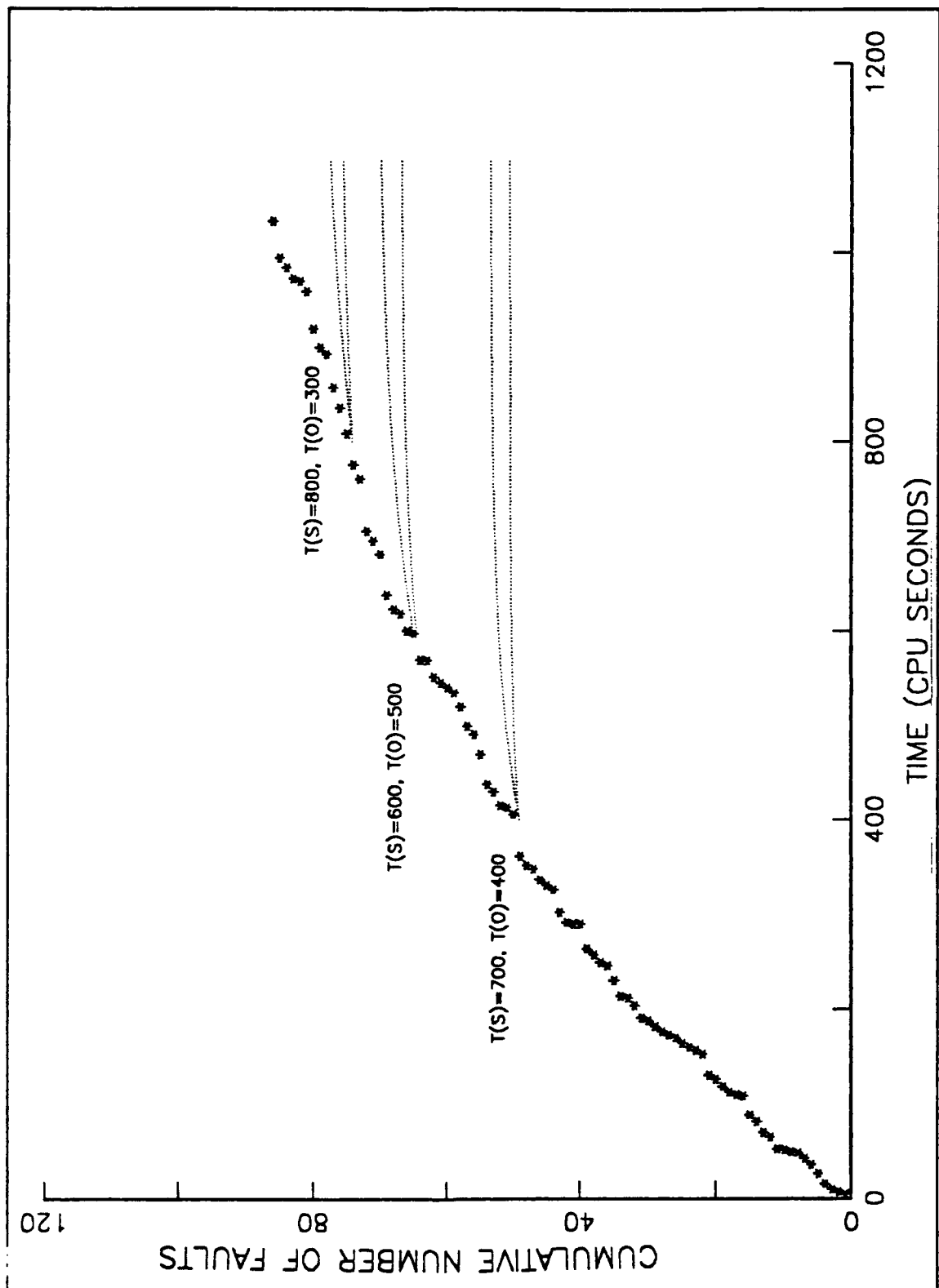


Figure A5. Data Set 2, $\Delta = 20$ (CPU seconds), Bayesian Method

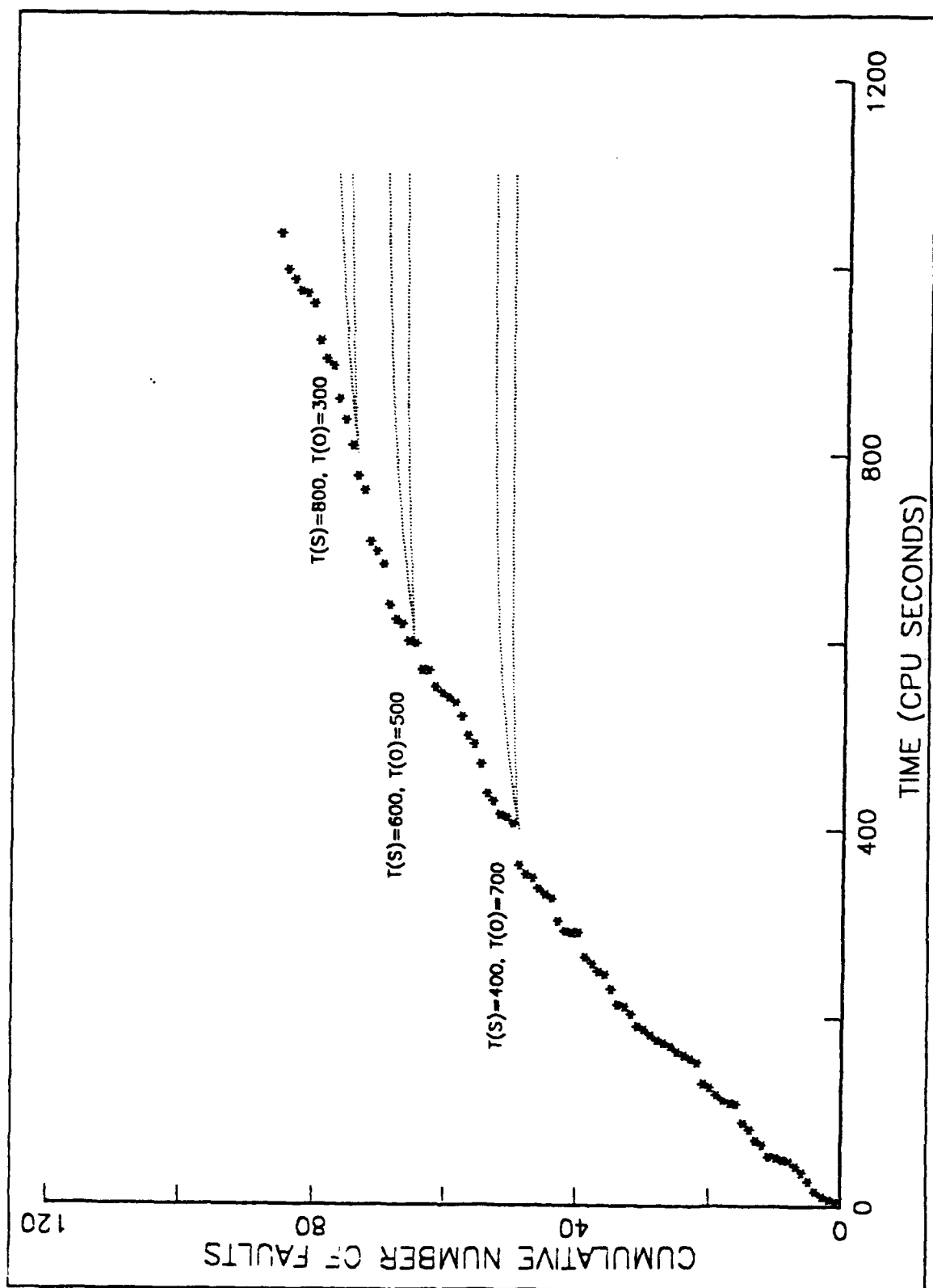


Figure A6. Data Set 2, $\Delta = 50$ (CPU seconds), Bayesian Method

REFERENCES

- Abdalla, Abdel-Ghaly, Chan, and Littlewood, B., "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, Vol.SE-12, No.9, pp.950-966, September 1986.
- Beizer, B., *Software System Test and Quality Assurance*, Van Nostrand Reinhold, 1984.
- Brooks, F., "No Silver Bullet," *Information Processing*, H.J. Kugler, ed., Elsevier Science Publishers, 1986.
- Dalal, S., Fowkles, E., and Hoadley, B., "Risk Analysis of the Space Shuttle: Pre-Challenger Prediction of Failure," *Journal of the American Statistical Association*, Vol.84, No.408, pp.945-958, December 1989.
- Dalal, S., Mallows, C., "When Should One Stop Testing Software," *Journal of American Statistical Association*, Vol.83, No.403, pp.872-879, September 1988.
- Dalal, S., and Mallows, C., "Some Graphical Aids for Deciding When to Stop Testing," *IEEE Journal, Selected Areas in Communications*, 1990.
- Efron, B., "Bootstrap Confidence Intervals," *Biometrika*, Vol.72, No.1, pp.45-58, April 1985.
- Farr, W., "A Survey of Software Reliability Modeling and Estimation," *Naval Surface Warfare Center, Dahlgren Virginia*, September 1983.
- Goel, and Okomoto, K., "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, Vol.R-28, No.3, pp. 206-211, 1979.
- Goel, A., "Software Reliability Models: Assumptions, Limitations and Applicability," *IEEE Transactions on Software Engineering*, Vol.SE-11, No.12, pp.1411-1423, December 1985.
- Hernandez, J., *Naval Postgraduate School Masters Thesis, "Derivation Strategy from Experience Based Test Oracles,"* 1989.
- Hetzel, B., *The Complete Guide to Software Testing*, 2nd ed., QED Information Sciences, Inc., 1988.

IEEE Standard Glossary of Software Engineering Terms, IEEE Press, 1984

Moranda, P., "Predictions on Software Reliability," 1975 *Proceedings of the Annual Reliability and Maintainability Symposium*, Washington, D.C., 1975.

Moranda, P., Jelenski, Z., "Final Report on Software Reliability Study," McDonnell Douglas Aeronautics Company, MDC Report Number 63921, 1972.

Moranda, P., Jelenski, Z., *Statistical Computer Performance Evaluation*, pp.465-483, edited by Frelberger, W., Academic Press, 1972.

Musa, J., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Company, 1987.

Musa, J., Okumoto, K., "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Bell Laboratories*, Bell Laboratories Technical Report, pp.230-238, 1984.

Schafer, R., Alter, J., Angus, J., and Enoto, S., "Validation of Software Reliability Models," *Rome Air Development Center Technical Report RADC-TR-79-147*, 1979.

Schick, G., and Wolverton, R., "An Analysis of Competing Software Reliability Models," *IEEE Transactions on Software Engineering*, Vol.SE-4, No.2, pp.104-120, February, 1978.

Schneidewind, N., "Analysis of Error Process in Computer Software," *Proceedings of the 1975 International Conference on Reliable Software*, pp.337-346, Los Angeles, California, 1975.

Shooman, M., "Operational Testing and Software Reliability Estimation During Program Development," *Record of 1973 IEEE Symposium on Computer Software Reliability*, IEEE Computer Society, N.Y. 1973.

BIBLIOGRAPHY

Allison, K. and Baca, J., "Software Maturity Processing Comes of Age," *Defense Computing*, July-August 1988.

Caruso, J., "Integrating Prior Knowledge with a Software Reliability Growth Model," *IBM Corporation*, 1990.

Dalal, S., and Mallows, C., "Buying with Exact Confidence," *Bell Communications Research*, 1990.

Department of the Air Force, AFOTEC Pamphlet 800-2 Vols 1-6, October 1990.

Dunn, R., Ullman, R., "Quality Assurance for Computer Software," McGraw-Hill Book Company, Inc., 1982.

Farr, W., and Srivastava, V., "The Use of Software Models in the Analysis of Operational Software System," *Institute of Environmental Sciences Proceedings*, 1985.

Farr, W., and Smith, O., "Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Library Access Guide," *Naval Surface Warfare Center, Dahlgren, Virginia*, March 1991.

Farr, W., and Smith, O., "Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide," *Naval Surface Warfare Center, Dahlgren, Virginia*, March 1991.

Humphery, W., *Managing the Software Process*, Addison-Wesley Publishing Company, 1989.

Littlewood, B., "Theories of Software Reliability: How Good Are They and How Can They Be Improved," *IEEE Transactions of Software Engineering*, September 1980.

McPherson, M. and Wiltse, J., "Software Maturity and Its Use as an Operational Test Readiness Criterion," *Technical Journal*, January, 1989.

Moranda, P., "Prediction of Software Reliability During Debugging," *Proceedings 1975 Annual Reliability and Maintainability Symposium*, 1975.

Ohba, M., "Software Reliability Analysis Models," *IBM Journal of Research and Development*, July 1984.

Ray, B., Bhandari, I., and Chillarege, R., "Reliability Growth for Typed Defects," IBM Watson Research Center, Yorktown Heights, 1987.

Siefert, D., "Implementing Software Reliability Measures," *The NCR Journal*, March 1989.

Wilson, L., and Shen, W., "Software Reliability Perspectives," unpublished researched going on at Old Dominion University, Norfolk, Virginia funded by NASA.

Yamada, S., Ohba, M., and Osaki, S., "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability*, December 1983.

Yang, M., "On Optimal Stopping Rules in Software Reliability," *Software Engineering Research Center*, 1991.

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142
Naval Postgraduate School
Monterey, California 93942-5002 | 2 |
| 3. | Department Chairman, Code 30
Department of Operations Research
Naval Postgraduate School
Monterey, California 93942-5002 | 1 |
| 3. | Professor Donald P. Gaver
Department of Operations Research
Naval Postgraduate School
Monterey, California 93942-5002 | 2 |
| 4. | Professor Timothy J. Shimeall
Computer Science Department
Naval Postgraduate School
Monterey, California 93942-5002 | 2 |
| 5. | Commander, Operational Test and Evaluation Force
Chief of Staff
Norfolk, Virginia 23511 | 1 |
| 6. | Commander, Operational Test and Evaluation Force
Technical Director, Code OTT
Norfolk, Virginia 23511 | 1 |
| 7. | Commander, Operational Test and Evaluation Force
Deputy Chief of Staff for OTE Support, 30 Division
Norfolk, Virginia 23511 | 1 |
| 8. | LT Thomas E. Dennison
1668 Toledo Court
Pacifica, California 94044 | 2 |
| 9. | Headquarters
Air Force Operational Test and Evaluation Center
LG5 Division
Kirtland Air Force Base, New Mexico 87117-7001 | 1 |